Accelerating Conservative Parallel Simulation of VHDL Circuits

THESIS

Joel F. Hurford
Captain, USAF

AFIT/GCS/ENG/94D-10

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/94D-10

Accelerating Conservative Parallel Simulation of VHDL Circuits

THESIS
Joel F. Hurford
Captain, USAF

AFIT/GCS/ENG/94D-10

Approved for public release; distribution unlimited

AFIT/GCS/ENG/94D-10

ACCELERATING CONSERVATIVE PARALLEL SIMULATION OF VHDL CIRCUITS

THESIS

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Engineering

Joel F. Hurford, B.S.C.S.

Capt, USAF

30 Nov 1994

Acknowledgements

This thesis is a product of the efforts of many individuals who have my gratitude. Many contributors are presented in the bibliography on whose shoulders I have stood. My thanks to current students in Parallel Computation at AFIT. These people can be found in the Parallel Lab around the clock, ready and willing to discuss theoretic implications, implementation techniques, available tools, or the last episode of the *X-Files*. I would like to thank the previous AFIT researchers of Parallel Simulation that built the tools that allowed me to conduct my study. Additional thanks goes to various and sundry faculty members at AFIT: Dr. Gary Lamont, LtCol Bill Hobart, Maj Roger Burk, Maj Mark Mehalic, and Dr. Dennis Quinn. These gentlemen taught me the foundations that directly supported my research. In addition, they availed themselves to yet another graduate student, me, pestering them with spurious and occasionally misguided questions. Of course, special thanks to my thesis committee that made all this possible. Dr. Henry Potoczny, my advisor, allowed me complete freedom to work through the issues of developing and constructing a thesis. He has my gratitude for his patience when schedules were missed and plans modified.

The individual that deserves great credit for the completion of this research is Dr. Thomas Hartrum. Advising many other students, this committee member met with me weekly to evaluate and critique progress. He was the scientific logic that formed the questions that required exploration. He was the antagonist that shook the structure of my research to test its integrity. He was the intellect that probed experimental evidence to verify my conclusions. He was the manager that kept my efforts proximate to the established schedule. If he cared to be a typist as well, his name would also appear on the cover page. I would feel more individual gratitude for Dr. Hartrum if I did not know that his generous assistance comes from his own passion and committment for the research topic. I regret that faculty are not offered a grade on their contributions to student research. If this were different, Dr. Hartrum has certainly earned his "A".

Joel F. Hurford

Table of Contents

List of Figures

Abstract

As modern digital circuits grow larger and more complex, the time required to perform sequential simulations becomes unacceptably slow. Since simulation is a vital input to the design and validation of circuits, this bottleneck affects the efficiency of the entire development cycle. Parallel simulation offers a solution that scales with the problem. By assigning circuit components to distributed processors, the work of the simulation can be divided. There is, however, an additional cost of synchronization between the cooperative processors not present in sequential simulation. The manner in which circuit components are partitioned among processors greatly influences the amount of overhead incurred. The task is to partition intelligently such that computational parallelism is not overwhelmed by synchronization overhead.

In this research effort heuristic techniques of intelligent partitioning were considered. By observing trends of successful partitions, a statistical relationship of *a priori*, graph-based parameters was developed with parallel simulation runtime. Formal definition of this relationship in the form of a cost model allowed allocations to be ordered by predicted runtime. By choosing the allocation with the lowest cost model value, the simulation using that allocation was expected to have the lowest runtime of the set considered. "The set considered" is an important distinction because the mapping of tasks to processors to achieve the lowest runtime is a known NP-Complete problem. Finding an optimal solution is intractable; finding relatively good solutions is desirable. The set of candidate allocations is chosen by using Kapp's AB Improvement search procedure (Kapp, 1993:84). Unfortunately, both current and previous cost models failed to achieve significant statistical correlation with runtime.

Improvement was achieved through controlling feedback. In example circuits, previous partitioning techniques induced feedback among processors. By eliminating this, better than linear speedup was achieved.

ACCELERATING

CONSERVATIVE PARALLEL SIMULATION

OF VHDL CIRCUITS

1. Introduction

1.1 Background

Simulation provides fundamental analysis capabilities for many theaters of research and industry. Similar to static models, simulations provide the ability to examine target systems at an economy of time and resources. Furthermore, simulations provide an ability to examine systems impossible to construct, for example weather patterns or sub-atomic particles. Simulations exceed static models by adding the ability to examine the dynamics of a system, not just the state of its components. Because of these benefits, computer simulation is a basic tool of defense, weather, biomedical, chemical, financial, electronic, and many other industries.

A repeating theme for the computer is its inability to keep up with user requirements. Each developmental leap of processing power is met by a larger need for computational resources. For simulations, researchers seek finer resolution of simulation, or consumption of larger datasets, or both. In the case of digital circuit design, chip transistor counts increase by 25% per year, doubling every three years (Hennessy and Patterson, 1990:17). For all the marvels of processor development, the sequential system creates an inherent bottleneck that will be realized with a sufficiently large problem or sufficiently small timing constraint. For circuit design, slow simulation lengthens the design cycle and increases the cost of the final product (Kapp, 1993:1). Assuming that computer users will continue to require more than computer designers can fit into a single data processor, a solution is to enlist more computers. Parallel processing is an alternative that may prevent simulation from being the limiting phase of the research or development cycle. Immediately, parallel simulation offers the ability to process larger datasets, perhaps in less time. More importantly (if scalable[1]) a large simulation can be accommodated by increasing the number of processors regardless of the size of the dataset. Or, by utilizing more processors, a solution to a defined problem can be found within some requisite timespan.

This paper results from the Advanced Research Projects Agency's (ARPA) desire to achieve performance improvement for the simulation of VHSIC Hardware Description Language (VHDL) circuits. ARPA sponsors the QUEST project with the objective of a thousand-fold speedup in large VHDL simulations (Kapp, 1993:1). Reasearchers at the Air Force Institute of Technology (AFIT) have been investigating conservative parallel simulation of VHDL circuits for several years. In 1992 Breeden (Breeden, 1992) demonstrated speedup for a random allocation of VHDL behaviors[2] to processors (See Figure 1). In 1993, Kapp showed the benefit of more intelligent partitioning strategies and the feasibility of iterative improvement of initial partitions using his AB Border Improvement process (Kapp, 1993:84). This study continues AFIT research by further exploring the potential of intelligent partitioning to increase speedup[3] of parallel VHDL simulation.

[1]Scalability is the ability to solve a problem for an increasing number of processors. Scalability is considered with regard to size, time, and memory consumption.

[2]A *behavior* is an executable VHDL process representing a logic gate, source signal, or other simple VHDL process (Kapp, 1993:2).

[3]*Speedup* is the ratio of single processor runtime / parallel runtime.

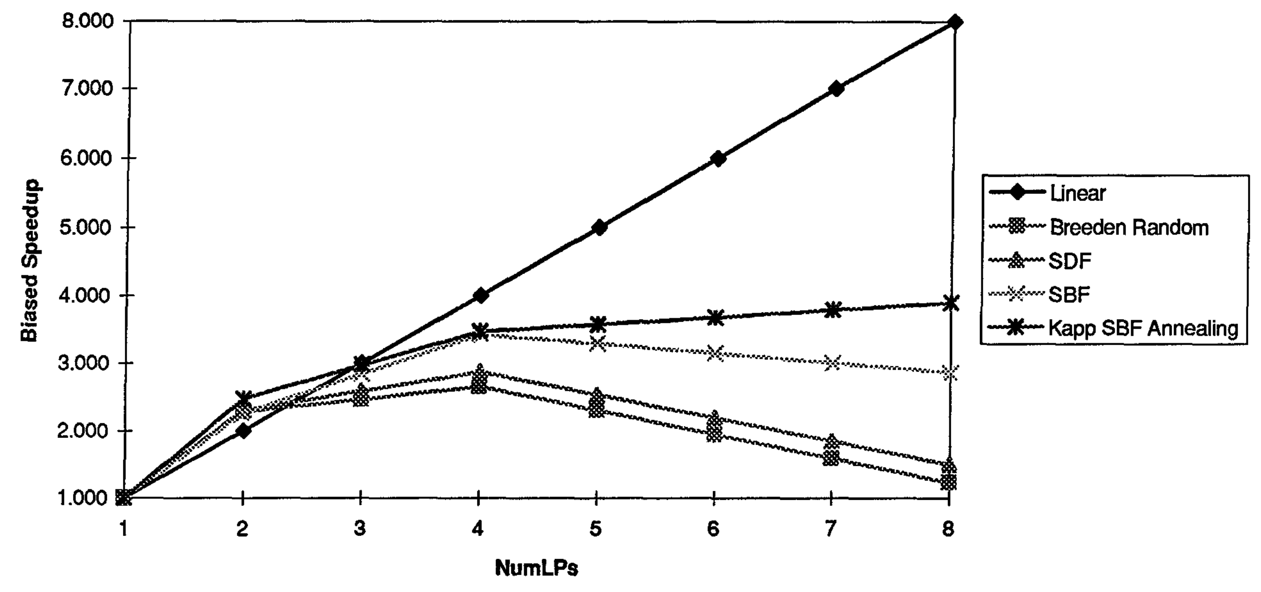


Figure 1 Speedup Curves for Wallace Tree (Kapp, 1993)

If the simulation of a VHDL circuit is the end problem, partitioning that problem for parallel execution is a preceding step of significant weight. Specifically, the allocation of *N* precedence constrained tasks to *P* processors to achieve minimum runtime is referred to as a version of the Mapping Problem and is NP-Complete (Sartor, 1991:1-4). Fundamentally, an NP-Complete problem has many potential solutions without a polynomial way to find an optimal in that solution space. Thus, it is unreasonable to seek an optimal solution for problems of large size (greater than 100 nodes to be allocated). Actually, problems of relatively small size are also unreasonable. Sartor gives the example of 60 nodes allocated to 2 processors. An exhaustive search of the 60! possible combinations would take 2.63 x 10^{66} centuries if a processor could consider 1,000,000 combinations a second. The circuits worthy of consideration for parallel simulation have more than 1000 nodes. Inherently, one is limited to a sub-optimal approximation of the best solution locatable in some reasonable amount of time. "Reasonable" need not be polynomial, but it does need to be amortizable over the number of simulation runs of the applicable circuit (Nandy and Loucks, 1993:46).

Furthermore, the goal of decreasing simulation runtime[4] cannot be measured without running the simulation. Ideally, one could determine the best partition using *a priori* parameters (measurable before the simulation). "However, it is often difficult to relate this goal (minimum runtime) to a usable parameter in the partitioning process" (Nandy and Loucks, 1993:43). The task evolves to find *a priori* indicators or a combination of indicators that correlate to a good runtime. Further complicating the task at hand is the number of parameters significant to simulation runtime. Candidate parameters include:

- number of behaviors
- number of interconnections
- number of dependencies crossing processor boundaries
- imbalance of load (number of behaviors) allocated to processors
- minimum path length of subgraphs allocated to each processor
- granularity[5] of host machine
- granularity of application
- fanout of behaviors
- level of feedback
- frequency of signal change
- efficiency of event list
- utilization of resources
- task switching capabilities of host operating system
- test bench size and nature
- many, many more

[4]In a parallel execution, *runtime* is the max[finish time of all processors] - min[start time of all processors].

[5]Hardware granularity refers to the computational capabilities of each processor versus the ability to communicate among processors. When computational and communication capabilities are equal, the machine is labeled fine-grain. Usually computational speed exceeds communication throughput making the machine tend towards coarse-grain. Software granularity is the ratio of the computational demands versus communication demands. Similar to hardware, low computational workload between communications indicates a fine-grain application. Greater computation to communication ratio indicates a coarse-grain application.

In order to make an analysis tractable, assumptions and scope must be used to focus the study and likely limit the general applicability of results (Bailey and Pagels, 1991:627).

Despite the daunting number of complications in parallel VHDL simulation, it is a greatly simpler case than general simulation since logic circuits have a static dependency graph. A logic gate is exactly wired at the start of a simulation as it is at the finish. Non-deterministic simulations allow actors to dynamically interact and consequently change lines of communication. Hopefully, conclusions from this research will find applicability in more general simulation cases.

1.2 Research Objectives

As alluded to in the previous section, the goal of this research is to speed conservative simulation of VHDL circuits. Pursuit of this goal includes graph based analysis of the subject circuit and statistical correlation of sub-graph parameters to runtime in order to achieve iterative improvement of a "first-cut" allocation. A *partition* is a component, or subgraph, of the subject circuit allocated to a single processor. An *allocation* is the set of partitions that make up a subject circuit and is the end result of the partitioning process including the mapping relation of partition to processor. *Iterative improvement* investigates moving candidate nodes from one partition to another assessing benefit based on a representative cost model. Small cost model results should correlate to comparitively small observed runtime values. Specific objectives are:

- Determine graph based partitioning strategies significant to the speed of VHDL simulations.
- Use statistical analysis to identify allocation parameters consequential to runtime.
- Form a cost equation that assesses the cost of an allocation and correlates to the runtime for that allocation. The cost equation should at least define a partial ordering of allocations.
- Demonstrate improvement of a statistically derived cost model over Kapp's theoretically determined model.

Questions to address in accomplishing the above objectives include:

- How general is the statistical cost equation? Is it circuit specific? Is it host architecture specific?
- How well does the statistical model do? How does it compare to other simple strategies (random, breadth-first, depth-first) and the Kapp Cost Model?
- How reliable is the statistical cost model? How frequently does it demonstrate benefit over the Kapp Cost Model? How accurately does it order allocations?

1.3 Assumptions

The toolset used in this research is based on the commercial Intermetrics VHDL compiler (Intermetrics VHDL Compiler, 1990). Previous AFIT researchers, Comeau and Breeden, developed VSIM which translates Intermetrics compiler output into parallel source code which can then be used to run on the host parallel computer (Comeau, 1991). VSIM requires SPECTRUM which provides the interprocessor communication services and implements a version of Chandy-Misra's null protocol (Chandy and Misra, 1988). VSIM has been executed on Intel iPSC/2, iPSC/860, and Paragon architectures. This report exclusively collects data on the iPSC/2. The final tool of significance is the Graph Partitioning Tool (GPT). Modified to verion 3.0, this software creates the necessary mapping files and delay information deduced from an allocation. GPT performs initial and iteratively improved allocations. There is a fundamental assumption that results from this toolset are applicable to other simulation environments.

A biased speedup metric is used to evaluate parallel circuit performance. Speedup is therefore overstated by using a non-optimized sequential simulation runtime in the numerator (Wieland, 1990:1). However, statistics gathered on single processor execution of VSIM demonstrate negligible execution of overhead routines which supports the assumption that biased speedup is representative of true speedup.

Modifications to the VSIM tool may affect runtimes of research conducted in this thesis. Thus, comparison to previous results will be accomplished via speedup. It is assumed that tool specific differences will be isolated allowing assessment of the statistical cost model.

Also presupposed is that observed data follows a linear relationship in the space defined by currently measured parameters. If runtime is very different than linear or pertinent parameters have not been included, the model will likely prove unacceptable except for the specific cases composing the sample set on which the model was derived.

1.4 Scope

As stated, the Mapping Problem is NP-Complete making an optimal allocation unlikely and unverifiable. This research seeks to achieve "good" allocations assessed by comparison with random and best-to-date speedup measures.

The focus is on the feasibility of effective *a priori* partitioning. Naturally the efficiency with which partitioning is accomplished is basic to the merit of that strategy. However, efficiency is not considered here under the assumption that once demonstrated as effective, additional research can be spent to make the allocation process efficient.

Only Chandy-Misra null protocol simulation of VHDL circuits is considered. Also, only VHDL simulation is studied. This reduces the scope to the static dependency subset of the general simulation problem.

In developing the cost equation, gates are assumed to be uniformly active. The test vectors used are general and do not seek to isolate circuit subcomponents nor boundary conditions. Additionally, despite the fact that runtime is a maximum measurement for some single processor of the simulation, data collected and analyzed is based on cumulative statistics across processors. It is assumed that individual processor performance is close to mean performance. This global view of the circuit allocation allows simpler analysis. Partitions are, therefore, constructed in a balanced manner to avoid deviation from this assumption. Consequently, potential advantages of non-uniform allocations are not investigated and specific data pertinent to the LP driving the maximum runtime is not reported.

Single circuit, multiple allocation ordering is pursued. It is rarely of interest to compare runtimes of two different circuits, only the runtimes of different allocations of the same circuit. Model development occurs for a single circuit. The general applicability of that model is considered separately.

1.5 Overview

Chapter 2 presents a current literature survey in simulation prediction and modeling including results from previous AFIT research. After establishing the basics of conservative parallel simulation and terms used in this paper, alternative cost models and significant parameters to runtime are presented. Chapter 3 discusses the methodology and practices followed in this study. The statistical analysis technique applied throughout is presented in detail. Other issues, including further detail of the tools used and specifics of the simulation environment are also furnished. Implementation specifics and data results can be found in Chapter 4. Chapter 5 presents the conclusions offered in this paper and recommends avenues of further study.

1.6 Summary

The simulation of VHDL circuits provides required insight for computer designers and is a necessary part of the circuit development cycle. Sequential simulation is inherently limited in that a problem can grow large enough to exceed the resources of that sequential system or violate constraints on execution time. Parallelization offers hope of a scaleable solution such that no matter what the problem size (or perhaps timing constraint), effective and efficient simulation can take place through the application of additional processors. Mapping is the NP-Complete problem that is coupled with determining an allocation. It seeks to partition a problem into subcomponents and to allocate each subcomponent to a processor to achieve some optimal condition. In this case, atomic behaviors of the overall circuit are grouped into partitions and allocated to processors to achieve short runtime. Since the mapping problem is so difficult, an optimum solution cannot be found within reasonable time for large circuits. This research uses statistical analysis of graph parameters to find good runtime performance. Effective models for simulation performance are still immature, "...analytic modeling of parallel simulation is an art, for which I hope to shed light (Nicol, 1991:30)."

2. Literature Review

2.1 Conservative Simulation

In order to proceed with the specifics of this research, a brief presentation of the actors and dynamics of conservative simulation is required. Fujimoto describes parallel discrete event simulation in his excellent article (Fujimoto, 1989). There are three major actors in a sequential dis-

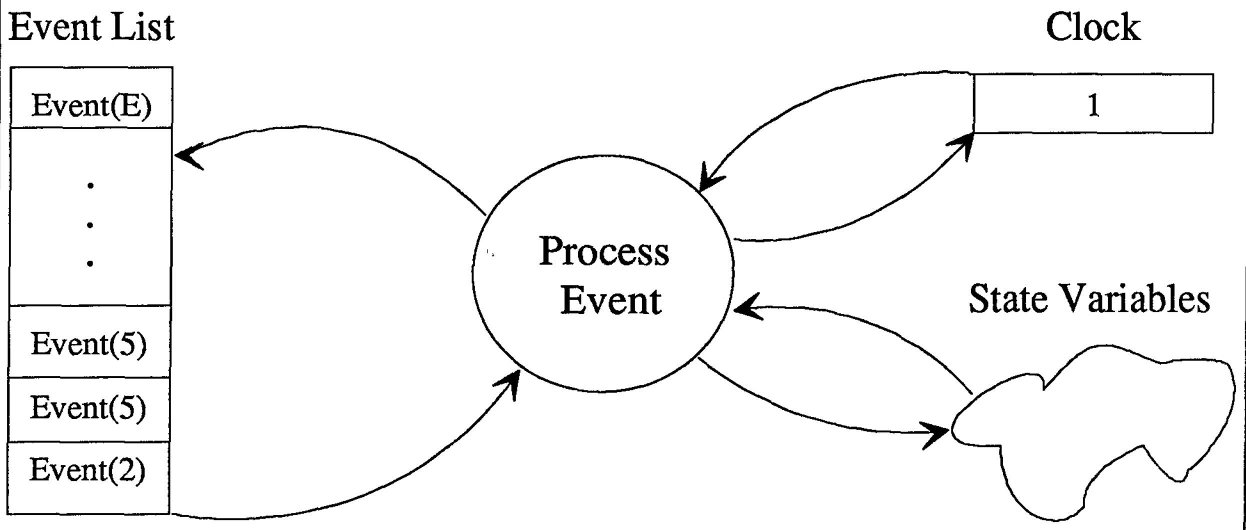


Figure 2 Basic Cycle of Discrete Event Simulation

crete event simulation as portrayed in Figure 2. A discrete event simulation proceeds by jumping from significant event to significant event as presented by the event list. Events are supplied for processing in monotonically increasing order of timestamp which represents the simulation time at which the event should occur. Processing an event requires updating the clock which tracks the progression of the simulation. Additionally, processing an event interacts with the state variables of the specific system being simulated. For VHDL simulation, state variables are specific segments of wire, or signals, which can be logical 0 or 1. Processing an event may cause future events to be scheduled as in the propagation of a signal change through affected gates. These new events are submitted to the event list which will hold them until all earlier events have been released.

Parallel simulation extends this basic sequential structure to multiple processors. As depicted in Figure 3, each processor has its own sequential simulator with each holding a disjoint view of the state variable space. In general PDES (parallel discrete event simulation) strategies avoid coherency issues by strictly forbidding direct access to shared state variables (Fujimoto, 1989:3). The collection of simulation structures and unique view of the state variable space de-

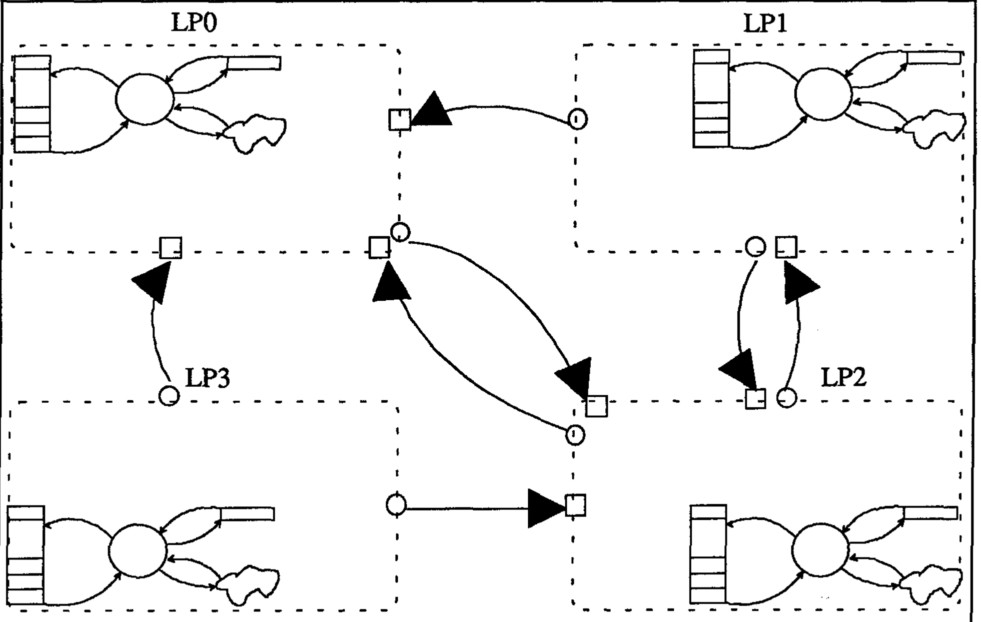


Figure 3 Basic Structure of Parallel Discrete Event Simulator

fines a Logical Process (LP). The $\bigcup LP_i$ for all i defines the state of the entire simulation (Fujimoto, 1989:4). Upon encountering an event that affects the state variable space of another LP, the source LP sends a timestamped event to the appropriate destination LP. Communication between LPs requires additional data structures. These structures may buffer or somehow prioritize message traffic, but the fundamental service required is to report to the owning LP the timestamp information of each event received (or sent) on the communication channel[1]. In addition to the communication data structures, some protocol must manage the presentation of remote messages to the local sequential simulator as well as the formatting of events destined for other LPs. It is the responsibility of the protocol to preserve causality.

Critical to any simulation is the concept of causality. A correct simulation does not allow causally related events to proceed out of order. If A affects B, A must be processed before B. A causality error occurs when events are executed out of order and amounts to the future affecting the past (Fujimoto, 1989:2). Two schools of thought prevail in discrete event simulation. Optimistic protocols use a *detection and recovery* method to identify when causality errors occur and rollback or otherwise correct the execution mistake. Conservative simulation uses *prevention* to avoid causality errors. Conservative protocols force each LP to block (suspend simulation) until it can be guaranteed that no messages from the past will arrive at the blocked LP. Chandy-Misra methods guarantee that messages sent over communication channels will be monotonically increasing by order of timestamp. Once all input channels have received a message equal to or later than an LP's local clock, that LP is guaranteed to be able to proceed with simulation without risking a causality error. Unfortunately, the act of blocking introduces the possibility of deadlock. Chandy-Misra's null message protocol uses messages that carry only a timestamp. The sender guarantees not to send any messages earlier than the null message timestamp. Null message recipients update their input channel and progress even though the sender has no real message for that LP. Chandy-Misra requires that all cycles of the contracted problem graph[2] carry a non-zero minimum delay and that nulls be sent:

[1]A *channel* represents a uni-directional line of communication between a source and destination. It is not a structure, but represents the possibility of communication between the two actors.

[2]The contracted problem graph, or *LP graph*, is the graph that results from contracting all nodes allocated to an LP into a single node.

- on all other output channels upon sending a real message down some channel
- on all output channels upon blocking
- on all output channels upon receiving a null message

Optimizations can be made to reduce null message traffic, but the above conditions guarantee to prevent deadlock. While deadlocks are prevented, cyclic dependencies are not. The propagation of null messages through a cyclic dependency allows all members to increase their input channel times and, consequently, their local clocks. The cycle in effect "winds up" to allow processing to continue on each participating LP.

2.2 Circuit Simulation

Circuit simulation is appropriate for discrete event simulation in that only a small fraction of logic gates, typically 0.1%, change output values on a single clock tick (Soule and Gupta, 1989:85). Synchronous or continuous simulation would leave a majority of processing resources idle in any time window. Soule and Gupta, however, state that this small activity level also precludes a null message protocol from being efficient. The implication is that real messages would numerically trail null messages by a similar ratio. Soule and Gupta are correct if using classic Chandy-Misra which assumes simple LPs and a large number of processors. The situation changes when logic gates are grouped onto LPs exceeding the simple processing envisioned in classic Chandy-Misra. Inter-LP arcs are collapsed into channels and only a fraction of logic gates on the LP border contribute to the generation of null messages.

Frequently, when developing cost models[3] for parallel simulation systems, researchers use queuing systems as the example application (e.g. Fujimoto, Nicol). Circuit simulation differs from queuing simulation in that queues do not generate new events. Every input event from a source in a queuing system realizes exactly one terminal event in some sink of that system. Circuit simulation differs in that fanout, cycles, and internal generators increase the event population of a circuit beyond the number of events generated from source nodes. Similarly, events can terminate at arbitrary vertices. If a signal change on the input of a logic gate does not change the output of that gate, no subsequent event will be scheduled. This makes it difficult to predict the population of events in the simulation at any given time which makes it more difficult to predict the processing activity of any circuit subcomponent.

2.3 Previous AFIT Results

2.3.1 Random Partitioning.

Breeden explored random partitioning which allocated behaviors to partitions arbitrarily, seeking only to maintain a balance in the quantity assigned to each LP. While very simple and quick to execute, this approach completely ignores the cost of inter-LP dependencies which are manifested from arcs[4] between behaviors on different LPs. The results in Figure 1 and other tests show that random partitioning provides speedup in limited cases and does not scale well due to overwhelming communication dependency overhead.

[3]A *cost model* is a mathematical assessment of some criterion of the simulation system. In this research, that criterion is runtime and the parameters of the mathematical model are graph based measurements of the partitioned circuit.

[4]A single signal between two behaviors is an arc.

2.3.2 Simple Data Partitioning.

Kapp first implemented Simple Data Partitioning (SDP) in his Graph Partitioning Tool v2.0. The problem graph is traversed in breadth-first or depth-first manner allocating behaviors to partitions as they are marked as found. Upon exceeding some threshold value for LP size, behaviors are allocated to the next LP. SDP algorithms are simple and quick to execute while yielding the benefit of allocating dependent behaviors to the same LP. Simple Depth-First (SDF) groups paths in a circuit onto an LP. This promotes larger minimum delays through LPs and around cyclic LP dependencies allowing null messages to "wind up" at larger increments. However, as shown in Figure 4, SDF with LP load balancing may result in fragmented paths which counteract the benefit of minimum delay. Simple Breadth-First (SBF) also succeeds in grouping dependent behaviors onto the same LP. Like SDF, SBF does not sufficiently counteract communication overhead particularly as the number of processors grows large.

2.3.3 Strongly Connected Components.

Strongly connected components (SCC) are a basic concept of graph theory and represent the largest set of nodes such that every node in a SCC is reachable by every other node. There can be many SCCs in a particular graph. SCCs can also be defined as a set of cycles. The SCC is the union of all vertices in inclusive cycles. Kapp and others observed the tightly coupled communication of cycle members in a problem graph. Even if the frequency of real message traffic between members is low, null protocol communication will have to occur within the SCC to prevent deadlock. Given this easy way to identify coupled behaviors, Kapp enhanced Simple Data Partitioning to consider SCCs as part of his AB Improvement partitioning technique described in the next section. The idea is to keep SCC members on the same LP to prevent their significant contribution to inter-LP communication.

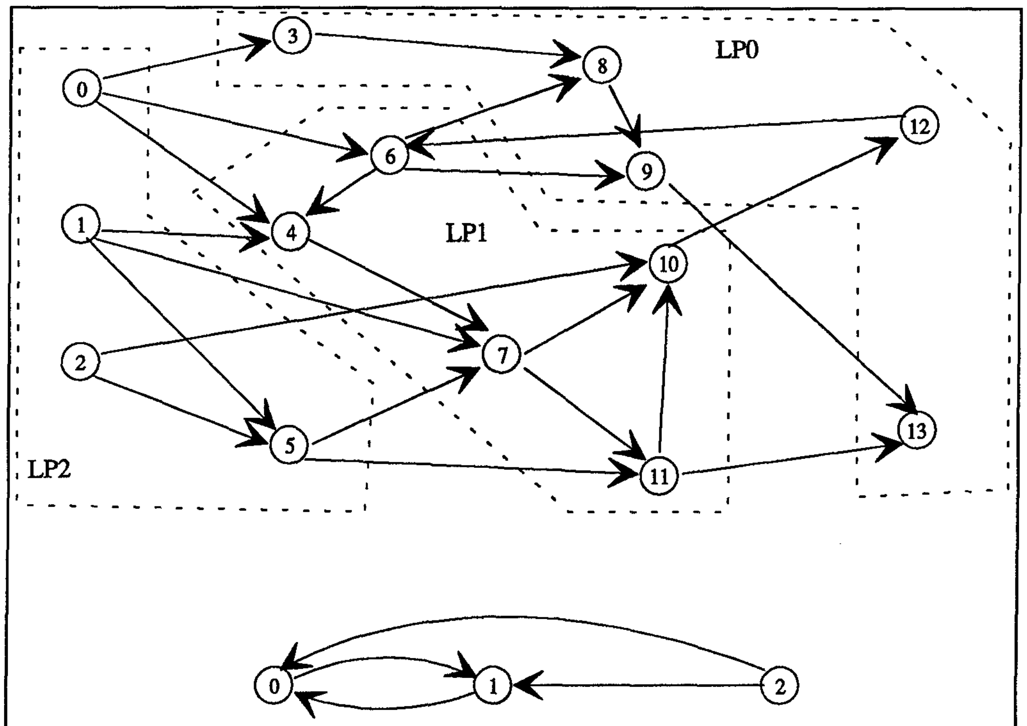


Figure 4 SDF Partition of Problem Graph & Resulting LP Graph

2.3.4 AB Annealing.

Despite the title, Kapp's AB Annealing process is actually an iterative improvement technique. Kapp performs a simple data partition with consideration of SCCs. Upon finding a node that is part of an SCC, all members of the SCC are marked *found* and allocated to the current LP under consideration. From this simple, efficient initial allocation, Kapp identifies a priority queue of candidate vertices for movement to other LPs. A candidate vertex is one that has more arcs to a single, external LP than to the LP to which it is currently assigned. A candidate vertex v has a priority calculated as follows:

Priority(v) = Max_External_Arcs(v) - Local_Arcs(v) (Kapp, 1993:61)

Each candidate vertex is considered for movement to all other LPs. If a move reduces the marginal approximation of communication cost and does not violate load balance tolerances, then the vertex is moved and the allocation data structures updated. A single iteration considers all candidate nodes. Subsequently, a new iteration begins by re-identifying candidate vertices and repeating the move processing. Iterations continue until a user-specified number or until iterations yield no improvement of the marginal approximation of communication cost. Kapp's entire cost model is

$$H := \beta \cdot H_n \cdot H_c \cdot \left(1 + H_d\right) + \alpha \cdot H_b$$

Equation 1 Kapp Cost Model (Kapp, 1993:57)

where

H_n : estimate for cost of null message traffic

H_c : estimate of cost for real message communication

H_d : estimate of effect of communication imbalance

H_b : estimate of effect of load imbalance

β : coefficient for communications cost

α : coefficient for processing cost

To estimate H_n, Kapp defines *Lookahead.* Lookahead is based on the minimum delays in traversing LPs. As stated, the magnitude of the minimum delay influences the overhead incurred in "winding up" cyclic dependencies and should therefore influence the number of nulls transmitted. Kapp defines $H_n = L_{arcs} \cdot O_{arcs}$ where L_{arcs} is Kapp's Lookahead and O_{arcs} is the number of communication channels in the LP graph. H_c estimates the amount of communication traffic induced by an allocation. Kapp forms a communication matrix C where each entry is

$$C_{ij} = \textit{number of arcs from } LP_i \textit{ to } LP_j \cdot \textit{weight of communication from } LP_i \textit{ to } LP_j.$$

Weight of communication between LPs allows consideration of the number of hops in the underlying architecture, available bandwidth on hardware communication line, etc. H_c is then found by

$H_c = \dfrac{\sum_i \sum_j c_{ij}}{total\ number\ of\ arcs}$. Kapp divides by the number of arcs to normalize this metric over different circuits. The distribution of communication is a unique term that attempts to assess the delay due to communication imbalance much like one would expect a delay due to computational imbalance between LPs. If one considers the maximum row sum for the communication matrix versus the average row sum for that matrix, a measure of communication imbalance can be formed. Let $D_i = \sum_j C_{ij}$ and $H_d = \dfrac{D_{\max} - D_{avg}}{D_{avg}}$. Finally, H_b adds the effect of computational load imbalance. By summing the computational weights of behaviors allocated in a partition, a measure for computational load can be assessed for that LP. Actual computational work also depends on the frequency of event generation for each behavior, but since that information not available a priori; only the static weight is considered. Similar to H_d, if

$Work_i = \sum_{behavior_s \in LP_i} weight\ of\ behavior_s$ then $H_b = \dfrac{Work_{\max} - Work_{avg}}{Work_{avg}}$. α and β are weighting parameters to balance the influence of communication costs versus computation cost for a specific host architecture. For the Intel iPSC/2, Kapp assumed $\alpha = 100.0$ and $\beta = 1.0$. Note that in the AB Annealing process, the entire cost equation is not evaluated to decide the movement of candidate vertices. Kapp instead uses $Cost = O_{arcs} \cdot H_c (1 - H_d)$. Summary results for all previous AFIT partitioning techniques on the Wallace Tree Multiplier are shown in Figure 1.

AB Annealing shows substantial improvement for the Wallace Tree at 8 nodes. However, this technique is suspect for several reasons:

1. Improvement requires finding a best simple partition. Ultimate speedup depends on the quality of the initial partition. Also, since this is a greedy technique, local minimums will limit the benefit of the iterative search.

2. Improvement for the Associative Memory is not as dramatic as for the Wallace Tree raising the concern of general applicability of the model.
3. Iterative improvement is based on a partial model thus losing the influence of some terms. If successful, the significance of those missing terms becomes questionable.
4. The layers of the model hide a canceling affect between H_c and H_d. H_c has a $\sum_i \sum_j C_{ij}$ term in its numerator and H_d has one in its denominator.
5. The estimates for α and β are arbitrary and require mathematical basis.

Despite these drawbacks, however, Kapp successfully demonstrated iterative improvement of run-time via graph based partitioning. In no instances did his model cause a worsening of the base allocation and does provide an efficient way to potentially get better allocations. It becomes the task of this follow on research to continue the successes of previous AFIT research and address its shortcomings.

2.4 Hierarchical Model

Chamberlain and Franklin studied the simulation of digital circuits on a hypercube architecture (Chamberlain and Franklin, 1990). Beginning with a simple cost model, they considered and expanded terms until measurable terms became available. Upon achieving the layer of measurable detail, they used their model to predict speedup. Unfortunately, the protocol used to manage the parallel simulation was based on a global synchronous clock and therefore cannot be directly applied to the distinct types of overhead incurred with a null protocol. However, the path of model development is applicable to any protocol.

Chamberlain's protocol uses a master processor to synchronize processing. Slave processors perform all computation of the simulation under the direction of the master. The master transmits a time to all slave processors which then simulate up to that global time (barrier) exchanging data with other slaves as necessary. Upon reaching that time, each slave sends a message indicating arrival at the global time and includes the time of its next event. After receiving arrival messages from all slaves, the master determines the lowest next event time and broadcasts that time to the slaves. Also fundamental to this cost model is the imposed assumption of hierarchical circuit structure. Circuits are built from components which are in turn built from sub-components which eventually are constructed from gates. In order for the synchronous protocol to be efficient, time steps must be sufficiently large to allow the processing of many gate level events. Thus, instead of

Table 1 Hierarchical Model Terms (Chamberlain and Franklin, 1990:11)

Variable	Type	Definition
R_p	Output	Simulation runtime using p processors
C	Input	Number of system components to be simulated
L		Number of levels in hierarchical system description
B		Number of busy ticks in the simulation
E		Number of events
F		Number of functional evaluations
M_∞		Number of state change messages when $P \rightarrow \infty$
k_l		Fraction of event/component evaluations at level l
α		Work distribution across communications links
β		Work distribution across processors
P	Design/Hardware specific	Number of processors
t_E		Single event processing time
t_{Fl}		Single functional evaluation time at level l
t_{CF}		CPU time for single message formulation
t_{CT}		CPU time for single message transmission
t_{CR}		CPU time for single message reception
t_{LM}		Link time for single message transmission
t_{LV}		Link time for single message protocol overhead
M_P	Abstract	Number of state change messages with P processors
H		Average number of hops required per message
W		Average communications width
t_{CPU}		CPU time per busy tick
t_{COMM}		Communications link time per busy tick
t_{SYNC}		Synchronization time per busy tick
t_{EVAL}		Event/functional evaluation time per busy tick
t_F		Average single functional evaluation time
t_{COMCPU}		Communications overhead for CPU per busy tick

allocating individual behaviors to LPs, Chamberlain and Franklin allocate by functional unit. Event times are based on functional unit delays, not gate delays. This is an example of increasing the granularity of the problem. Unfortunately, increasing the granularity decreases the potential degree of parallelism. From this simulation system, Chamberlain and Franklin build their model.

If the simulation executes for B busy ticks (a busy tick is a clock increment in which simulation activity occurs),

$$R_P = B \cdot [\max(t_{CPU}, t_{COMM}) + t_{SYNCH}]$$

This assumes that computation and communication occur concurrently in a busy tick. CPU time dedicated to message formation and protocol service are included in a component term of t_{CPU}. t_{COMM} refers to the time spent in inter-slave communication for cooperative processing. t_{SYNC} is the communication with the master for global synchronization. The CPU time used over all processors per busy tick can be broken down as $t_{CPU} = t_{EVAL} + t_{COMCPU}$ where t_{EVAL} is time spent performing actual event/functional evaluations and t_{COMCPU} is CPU time spent forming and executing message protocol. The CPU time spent evaluating can be expressed in terms of the time to perform an event evaluation per busy tick, average number of events per busy tick, time to perform a functional evaluation, and average number of functional evaluations per busy tick. On a per processor basis, this becomes $t_{COMCPU} = \beta \cdot [(E / BP)t_E + (F / BP)t_F]$. β is a imbalance coefficient since the computational work may not be completely uniform. The average time to perform a functional evaluation is the sum of all functional evaluations divided by the number of evaluations. On average there are M_p/B state change messages over the communication network between interacting slaves. On average, each processor receives M_p/BP messages on each busy tick. Messages travel through an average number of intermediate nodes which must incur processing time to forward them. The resultant expression for CPU time spent in communications overhead is

$$t_{COMCPU} = \alpha \cdot \{H \cdot (M_P / B)[(t_{CT} + t_{CR}) / P] + (M_P / B)(t_{CF} / P)\}$$

α is a term representing communications imbalance and serves a similar purpose as β. t_{COMM} considers the expected volume of inter-slave messages versus the available bandwidth of the architecture as

$$t_{COMM} = \alpha \cdot H \cdot (M_p / B) \cdot [(t_{LM} + t_{LV}) / W]$$

The last term t_{SYNC} is assumed to be accomplished via a complete exchange. In a hypercube architecture, each processor sends a message down $\log_2 P$ adjacent channels.

$$t_{SYNC} = t_{LV} \cdot \log_2 P + t_{LM}(P-1)$$

A version of this model was used to simulate several circuits allocated by a simulated annealing partitioning strategy. The results were disappointing in that random allocation consistently exceeded the performance of the simulated annealing allocation. Chamberlain and Franklin cite the following reasons for this unexpected result:

- The cost model does not consider load balancing while that is all the random partition considers -- load balancing is a pertinent parameter.
- The frequency of evaluation was assumed uniform over all components. Some components were executed 12 times more than the average causing a significant computational and communications imbalance.
- Achievable parallelism is limited by the global clock protocol to the number of processors able to simulate at any single point in simulation time.

Additional problems with this model include the high degree of knowledge required of a simulation and its environment. Some terms can only be determined by running the simulation once and feeding that information back into the cost model. Many terms are average quantities that would suffer the same failure as assuming uniform activation frequency for behaviors. Significant deviations from the average are probable and will throw the model. While the terms themselves are questionable, the way in which they were identified and determined are admirable. Any model development must reflect the theoretical basis of the system in meaningful terms; otherwise the model becomes incomprehensible and unextendable.

2.5 Parameters of Partition Models

An early question in model development is, "which parameters are important?". Failure to identify a robust set of candidates will ignore pertinent degrees of freedom (dimensions) of the target space and limit the correctness of the model to a subset of that space. What parameters are important in predicting the runtime of conservative VHDL simulations? Certainly, the number of gates is influential. Also, the size and nature of the test vector is material. This section presents additional parameters of merit.

2.5.1 Lookahead

The minimum delay through an LP of the simulation allows the prediction of the earliest subsequent event to come from that LP. Special considerations must be given to event generating applications like VHDL simulation which will be discussed later. "Quantitatively, if a process has knowledge of all events that will occur up to simulated time *T*, and can predict all new events it

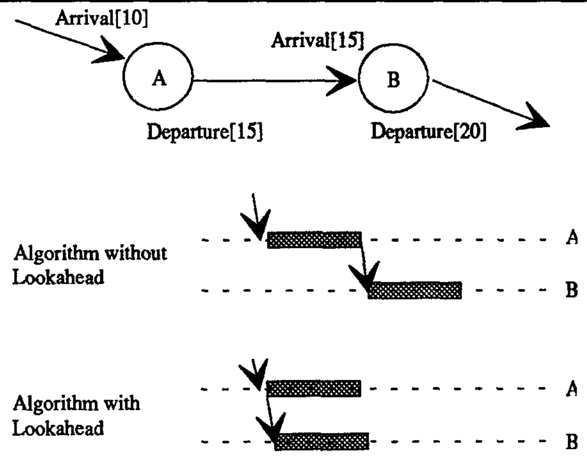


Figure 5 Queuing System and Lookahead

will generate with timestamp $T + L$ or less, then a process is said to have lookahead L" (Fujimoto, 1988-1:34). A lower bound on lookahead is the minimum delay through that process. By the definition, additional knowledge (perhaps event dependent) could allow larger lookahead instances. But this would require more intelligent processing. Minimum delay is static for static problem graphs and is generally synonymous with lookahead used in practice. Reducing lookahead limits the parallelism inherent in the system. For example, in Figure 5, Queue A sends an arrival event to B for each local departure event. If the processing time of each queue is known (say 5), greater

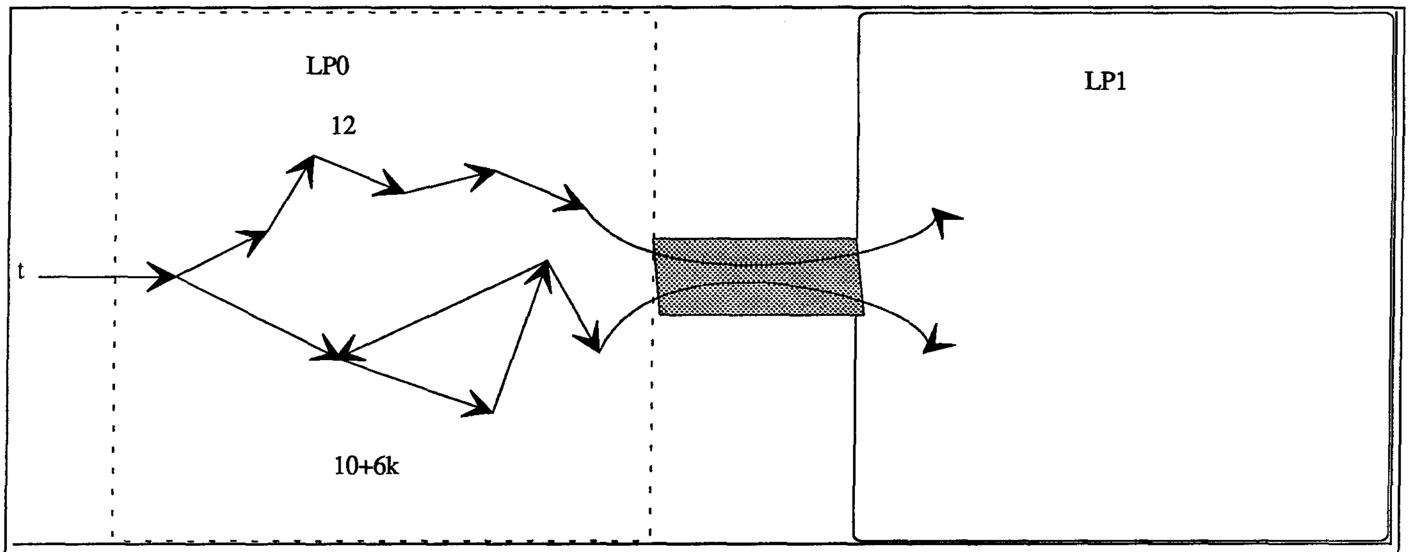


Figure 6 Multiple Paths and Lookahead

parallelism can be exploited by immediately sending B its arrival for T+5. This is an overly simplified example since Queue A may have jobs in its queue and cannot process the new arrival for some time. However, this, too, is a known quantity. Instead of sending *arrival_time + processing_time*, A can send an arrival to B at *LastScheduledFinishTime + processing_time*. Conservative simulations are acutely sensitive to cyclic dependencies. Expanding the previous single queue example to the LP level, lookahead is critical to the speed at which cycles are wound up and simulation allowed to proceed.

As previously mentioned, circuit simulation differs from queuing simulation in that events can procreate or terminate between source and sink nodes. Multiple paths through an LP may cause events to arrive at a destination LP at a time increment less than the minimum delay through the LP. As shown in Figure 6, the upper path is 12 hops and the lower path is minimally 10 hops. Consequently, any event arriving at LP0 will not be seen by LP1 until at least $t + 10$. It would be beneficial to allow LP1 to simulate up to this time. Unfortunately, a single event to LP0 may generate many arrivals at LP1. An event at t may generate arrivals to LP1 at times t+10, t+12, t+16, t+22, ... This is caused by multiple paths and an included cycle which may periodically generate events indefinitely. Thus, an arrival at LP0 at t does not free LP1 to simulate up to $t + 10$. A previous arrival at LP0 may have caused events destined for LP1 at times less than $t + 10$.

This does not completely deny the benefit of lookahead to digital simulations, it just requires more intelligence in determining lookahead. LPs are informed of lookahead possibilities via null messages. Upon any arrival, the source LP adds its minimum delay to the arrival timestamp and sends an appropriate null to the destination LP. As discussed, minimum delay is not guaranteed to represent the minimum time increment for a departure from the source; the source must test for earlier departures. A simple way to do this is to check the next event list of the local sequential simulator. The time of the earliest event on the list is a lower bound on potential departures. Thus, a safe time to send adjacent LPs is $\mathbf{min}[t_{next\ event}, t_{clock}+delay]$. The first term of the min function is extremely limiting since many events will be in the next event list with timestamp less than t_{safe}[5]for the destination LP. However, it is hoped that the later term will be used with enough frequency to yield the benefits of lookahead. Figure 7 demonstrates that as the number of LPs grow (and com-

[5]All incoming events to an LP are guaranteed to be later or equal to t_{safe}. Thus, the LP may safely progress to that point.

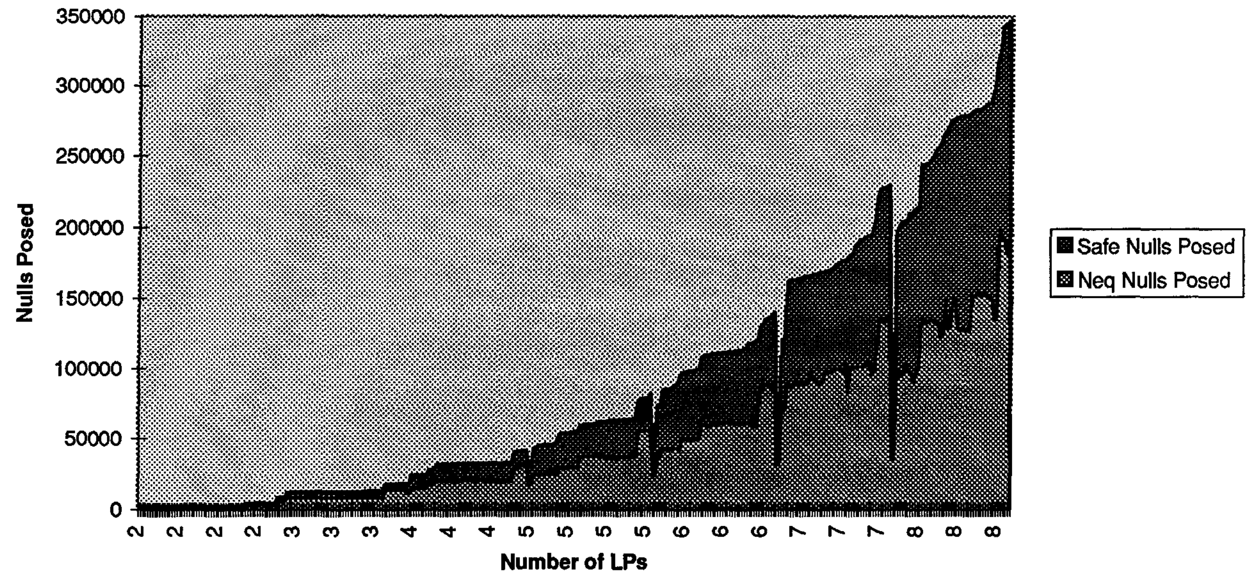


Figure 7 Nulls Posed with Delay and Event List Timestamps (Wallace Tree)

putation becomes finer grained), the number of null messages posed for transmission with the minimum delay timestamp grows relative to those posed with the minimum event list timestamp.

2.5.2 Lookahead Ratio.

Fujimoto added to the definition of lookahead by observing that the absolute value for lookahead is not as important as the lookahead relative to the time between the arrival and departure of an event. Similarly, Wagner and Lazowska observed that "since LPs affect each others' clocks by exchanging messages, this implies that lookahead values need to be comparable to the average message timestamp increment in order to be useful" (Wagner and Lazowska, 1989:150). Intuitively, a lookahead of x is somewhat meaningless unless x is large relative to the expected delay through the circuit. Fujimoto, therefore, defines Lookahead Ratio (LAR) as

$LAR = \frac{mean\ timestamp\ increase}{lookahead}$ (Fujimoto, 1988-2:18). Note that this formula assumes that lookahead is constant. In reality, lookahead can be unique for all (source LP, destination LP) pairs. Furthermore, Fujimoto offers no advice on an aggregate LAR metric that incorporates the LAR values for all LPs of an allocation.

2.5.3 Contention

Increasing the number of processors upon which a circuit is simulated makes the application finer grained (less average computation between communications). Correspondingly, communications overhead will likely increase as will the influence of contention. Contention occurs when message delivery between processors is delayed due to unavailable physical channels of the underlying hardware. Chittor and Enbody develop a mathematical model to predict actual message throughput (λ_p) based on an applied message rate (λ_a) and a saturation message rate (λ_{sat}).

$$\lambda_p^2(1-\lambda_a)-\lambda_p(\lambda_a+\lambda_{sat})+\lambda_a\cdot\lambda_{sat}=0$$ (Chittor and Enbody, 1991:II-3)

This model, like others, requires accurate prediction of the message generation rate of LPs which is very difficult to come by for logical circuits. Chamberlain has proposed pre-simulation to form reasonable estimates of circuit activity.

2.5.4 Fanout.

Nicol observed that the higher the fanout, the more descendent nodes are held back by conservative blocking. This is particularly true of fine grained applications. Grouping nodes together onto an LP as well as partitioning to increase lookahead will serve to diminish the amount of blocking occurring in the simulation (Nicol, 1991:34). Then, Nicol maintains, for coarse grained machines, performance degradation is not due to blocked processors, but due to communication and synchronization overheads (Nicol, 1991:42).

2.6 Techniques and Special Considerations

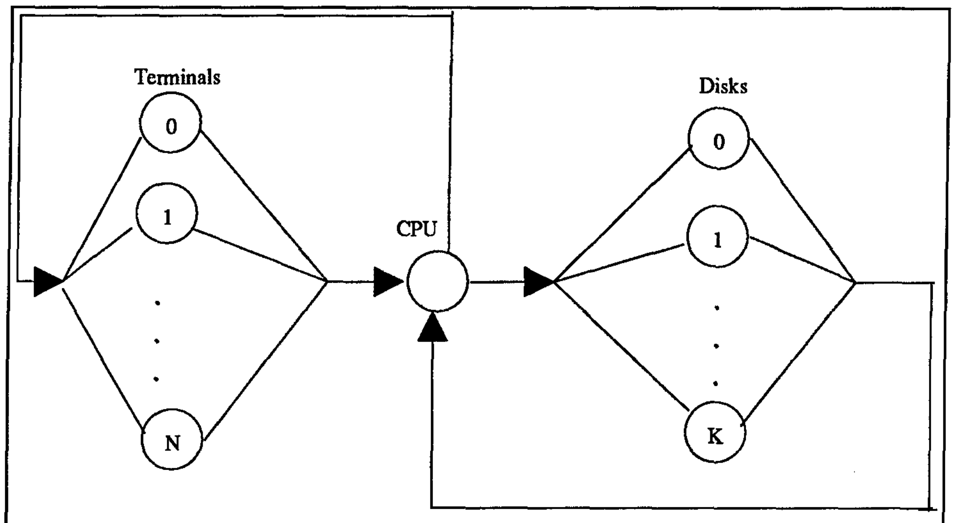


Figure 8 Bottleneck of Queuing Model

Bottlenecks in the problem graph present inherit limitations to parallelism that may not be observable in the actual system being simulated. Wagner and Lazowska give the example of a queuing network model of computer terminals accessing a central CPU with multiple disks attached (see Figure 8). In an actual system as depicted, the service time of the CPU would be much less than the service time of the disks such that the CPU would not delay disk service requests. In a simulation, however, a CPU service and disk service are of the same relative magnitude. Ideally, all nodes of Figure 8 would be simulated on unique processors. However, since the disks receive all requests from the CPU and the maximum utilization of the CPU is 1.0, the sum of disk utilization cannot exceed 1.0. Nor can the terminals' effective utilization exceed the consumption rate of their only destination. Thus, the maximum parallelism available in this model is 3.0 (Wagner and Lazowska, 1989:147).

A fundamental structure of discrete event simulation is the next event list (also imprecisely called the next event queue). This structure will hold all the events generated and processed on an

LP. Its efficiency is material to the speed of processing. Nicol recommends a splay tree implementation for large event populations (Nicol, 1993:325).

2.7 Statistical Analysis

"The goal of scientific analysis is the collection and organization of information concerning the world around us with the goal of increasing our understanding of the things we observe." (Thorndike, 1978:3). Thorndike continues to outline the stages of scientific discovery: observation, organization and prediction, explanation and understanding. This study fits in the second stage of "exploration". While there are many parameters with documented relationships to simulation runtime, many others remain with inter-relationships that are currently unexplored and undefined. Using basic techniques of statistical analysis, further insight to parallel simulation modeling is sought.

Numbers are a convenient way to label parameter values. Despite the absolutism of digits, numbers do not always carry the same meaning. When measuring parameters of interest, four types of scaling are pertinent. *Nominal* scaling attempts no quantitative assessment; it merely identifies membership to a particular category (e.g. male, female). *Ordinal* scaling makes general quantitative implication within a category (e.g. IQ levels). *Interval* scaling provides a precise quantitative relationship between differing values within a category (e.g. degrees Celsius). *Ratio* scaling is the final type and is interval scaling where 0 represents the absence of that trait in the sample (e.g. degrees Kelvin, or age).

The principle of least squares forms the basis of correlational procedures. Finding a least squares solution to a problem asserts that any other point, line, or curve will result in a larger sum if the difference between the estimator and each measured result is squared and summed together. It provides the most accurate predictor "on average for the group" (Thorndike, 1978:21). It does this regardless of the shape of the observed data. However, if the data is decidedly non-normal, other statistics may be better suited for the analysis. Each parameter will vary over observed data points. If not, no significant conclusion can be made about the parameter and the criterion to be predicted. If a parameter increases as the criterion increases, there is a positive correlation. If a parameter change of amount x consistently corresponds to a criterion change of amount y, then a high degree of correlation can be expected between the parameter and the criterion. Variance of the criterion can therefore be explained by variance in the parameter. The ratio of explained variance (SS_R) to unexplained variance (SS_E) indicates the significance of the model, or the ability to conclude that the predictor is not coincidentally related to the criterion. The total variance (SSy) is the sum of explained and unexplained variance.

Particularly important to the accuracy of a statistical model is the assumption of linearity. If the data samples fail to follow a linear form, than the use of a linear model is inappropriate. The difficulty becomes "seeing" if the data is linear in k-dimensional space. Linearity of two dimensional cross-sections (partials) of the overall sample space does not imply linearity of the problem. Based on a sound theoretical model, mathematical techniques can be used to form a linear model. Alternatively, polynomial and logarithmic regression techniques are available. A linear regression equation has the following form:

$$y = \beta_0 + \beta_1 \cdot x_1 + \ldots + \beta_k \cdot x_k$$

Equation 2 General Linear Equation

or, in matrix form,

$$y = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \cdot \\ \beta_k \end{bmatrix} \bullet \begin{bmatrix} 1 & x_1 & \ldots & x_k \end{bmatrix}$$

or

$$y = \vec{\beta} \bullet \vec{x}$$

Equation 3 Matrix form of General Linear Equation

Equation 3 represents the prediction of a single criterion value *y*, given a parameter vector $\vec{x}$ and a coefficient vector $\vec{\beta}$. Each entry of the parameter vector, x_j, represents a single sample measure for some unique parameter. For sample *i* of the sample set,

$$y_i = \vec{\beta} \bullet \vec{x}_i$$

Let *X* be a matrix such that row *i* of $X = \vec{x}_i \ \forall i$. This two-dimensional matrix has each row as the vector of parameter values for a single data sample. Each column is the vector of values for a single parameter over all samples. Using the vector $y_i \ \forall i$ as the predicted results, sum of squares comparison to observed data is possible as is correlational and significance testing.

Alternatively, observed *y* values can be substituted in the above equation to find the $\vec{\beta}$ that results in the minimum sum of squares. Since the rest of the discussion uses only the vector and matrix forms of variables, let $\beta = \vec{\beta}$, x = *X*, and y= vector of observed runtime values. As such, the following equation finds β:

$$\beta := \left(x^T \cdot x\right)^{-1} \cdot x^T \cdot y$$

Equation 4 Coefficients of Linear Model

Sum of square error not explained by parameters can then be found by:

$$SSe := y^T \cdot y - \beta^T \cdot x^T \cdot y$$

Equation 5 Residual Error of Sum of Squares

Sum of squares explained by regression is:

$$SSr := \beta^T \cdot x^T \cdot y - \frac{\left(\sum_{i=0}^{n-1} y_i \right)^2}{n}$$

Equation 6 Regressive Error of Sum of Squares

To test the significance of the model, the appropriate F statistic is calculated by:

$$F_0 = \frac{SS_R / k}{SS_E / (n-k-1)} = \frac{MS_R}{MS_E}$$

$$\textit{reject insignificance if } F_0 > F_{\alpha,k,n-k-1}$$

Equation 7 Linear Regression Test of Significance

where *k* is the number of parameters (columns) in the *x* matrix, and *n* is the number of samples taken (rows of *x* matrix).

If a model has already been established, statistical testing is straightforward using the above equations. If no model exists, one must be formed from the candidate set of parameters. Many techniques are available to construct a linear model. Thorndike proposes starting with the candidate with the highest correlation. Add to this the parameter which results in the greatest increase of the F_0 statistic. Test each parameter of the combined model for significance by a new statistic $F_P = \frac{(SS_R - SS'_R)}{MS_E}$ where SS'_R is the regressive sum of squares of the linear model excluding the parameter of interest. Essentially, this test sees if the marginal contribution of the parameter of interest is significant. If the marginal test indicates insignificance, remove that parameter from the model. Continue building by the F_0 test.

It is difficult to assess the general applicability of a model built from sample data. That sample data may be biased or insignificant to the characteristics of the overall population from which the samples are drawn. Thorndike recommends a double cross-validation technique to limit this risk. The sample set is divided in half and independent models developed for both halves. If each independently developed model appears similar to the other in form while effectively predicting the other's sample data, then the parameter set of the model is verified and can be *redeveloped on the combined sample.*

2.8 Summary

Conservative parallel discrete event simulation carries with it assumptions of the basic data structures used. Conservative simulation guarantees that an LP will never receive a remote event with a timestamp less than its current clock. It does this by blocking an LP until all input channels have a most recent communication timestamp greater or equal to the next event time of the LP. Blocking, however, creates the risk of deadlock which can be eliminated with the use of null messages. Previous AFIT researchers succeeded in achieving limited speedup of circuits using random, simple data, and iterative improvement partitioning strategies. Key to hill-climbing or annealing searches is a cost function to evaluate the worth of the current search position. Chamberlain developed a model for a synchronous protocol simulation and others like Nicol, Fujimoto, and Wagner have identified parameters of interest to any cost model. No published cost model has been demonstrated as statistically significant to partial or total ordering by runtime of VHDL circuit simulation. One particular difficulty with VHDL is the inability to predict load balance since nodes may create or destroy events differently depending on the input vector. Using statistical analysis may offer some insight into model building and add the legitimacy of mathematical significance to proposed models.

3. Methodology and Tools

3.1 Overview

The approach to this research is outlined by the steps of statistical analysis discussed in Section 2.7.

1. Using the Graph Partitioning Tool (GPT) from Kapp's research, generate a sample set of Wallace Tree simulations.
2. Use this sample set to form a cost model using double cross-validation.
3. Simulate the Wallace Tree to create a validation set to test the reliability of the statistical cost model.
4. Simulate the Associative Memory circuit to test the general applicability of the statistical cost model.

Support of this plan required major revisions to the Graph Partitioning Tool (GPT). Also, addi-

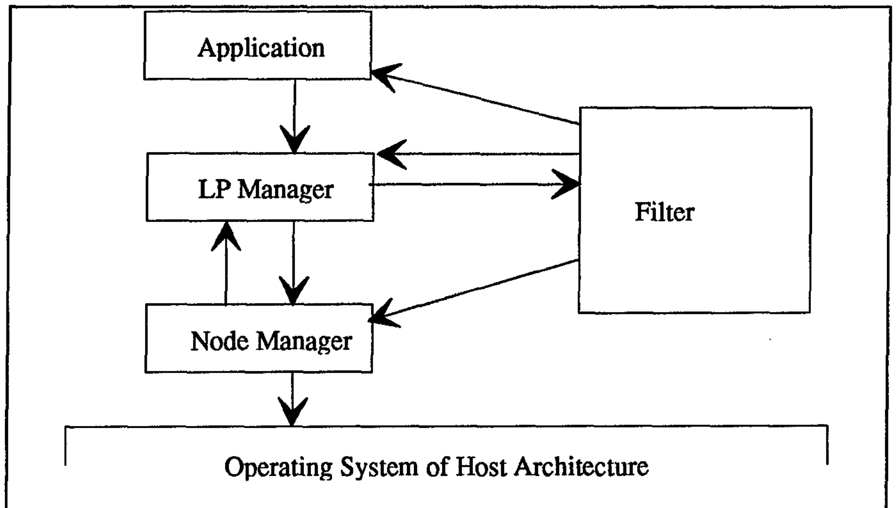


Figure 9 SPECTRUM Structure and Internal Dependencies

tional instrumentation of SPECTRUM provided insight into the work breakdown of processors in VHDL simulations.

3.2 Spectrum

SPECTRUM was originally developed at the University of Virginia on a BBN Butterfly. AFIT researchers converted this code for execution on the Intel iPSC/2 hypercube. SPECTRUM uses a SPMD (Single Program Multiple Data) model which means that each processor has its own copy of the simulation. The state space is divided among processors using the configuration files

Table 2 SPECTRUM Standard Filter Descriptions

Filter Name	Access Control
Initialization	Invoked as part of simulation startup, this fills an array with the addresses of filter functions for the six defined filters.
Get Event	Upon a request by the application for the next event to process
Post Event	Upon a request by the application to send a specified event to some remote destination
Post Message	Upon a Node Manager request to provide a remote event to the local LP Manager
Enqueue Event	Upon needing to place a specified event into the next event list
Advance Time	Upon needing to update the LP Manager clock
Termination	Upon conclusion of the simulation usually to collect and print statistics

lp*x*.arcs and lp*x*.map where *x* is the number of LPs. Figure 9 shows the structure of SPECTRUM which uses various levels of abstraction to simplify and standardize the interface to interprocessor communication for the application. As has been demonstrated at AFIT, applications using SPECTRUM have been ported to other architectures with little or no recoding of layers above the node level interface. SPECTRUM claims compatibility with any synchronization protocol and does not innately favor conservative versus optimistic techniques. In other words, SPECTRUM does not implement a protocol which must be developed separately. It does, however, provide a structure upon which a protocol can be built. The synchronization protocol is implemented in the filter layer of SPECTRUM. Well defined points of execution allow filter routines to be called and thereby control the communication between LPs. As can be seen by the dependencies of the various "layers" of SPECTRUM, the LP Manager, Node Manager, and Filter are tightly coupled. The Filter layer is particularly complicated with its access to (and knowledge of) all layers but the operating system. Overall, the application benefits from a defined, simplified interface to SPECTRUM and SPECTRUM benefits from a defined localized interface to the operating system.

The interaction of the SPECTRUM abstractions are important. Only LP Manager directly calls filter routines which are "registered" with the LP Manager as a part of the initialization of the

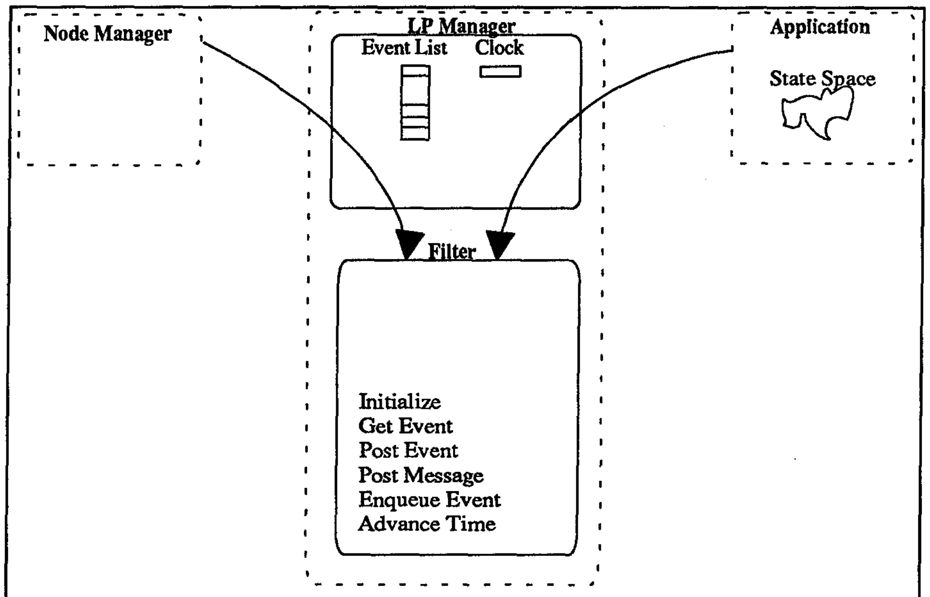


Figure 10 Redirection of LP Manager calls to Filter

system. Those routines and invocation points are described in Table 2. As visible in Figure 10, the basic discrete event simulator is broken between the application and SPECTRUM. LP Manager controls the next event list and a clock reflecting the time of the last message processed. The application performs the interpretation and consequence of events which it applies to its state space. When the application requires an event to process, it requests it from the LP Manager. If a Get Event filter is defined, LP Manager redirects the request to the Filter layer. The behavior of the Get Event filter is protocol dependent. For a null message protocol, Get Event will block until it is able to find an event in the next event list with a timestamp earlier than or equal to the safe time. This may require waiting until a remote LP sends an event with the appropriate timestamp or a remote LP sends a message that allows updating t_{safe}. Similar redirection of LP Manager calls to the filter occur for the other filter functions.

3.3 VSIM

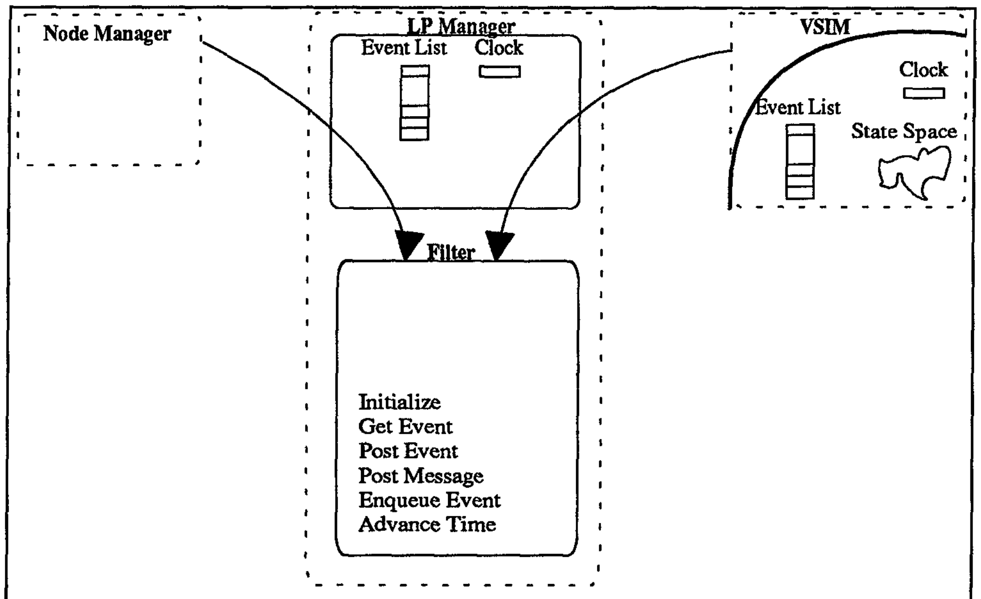


Figure 11 VSIM Interaction with SPECTRUM

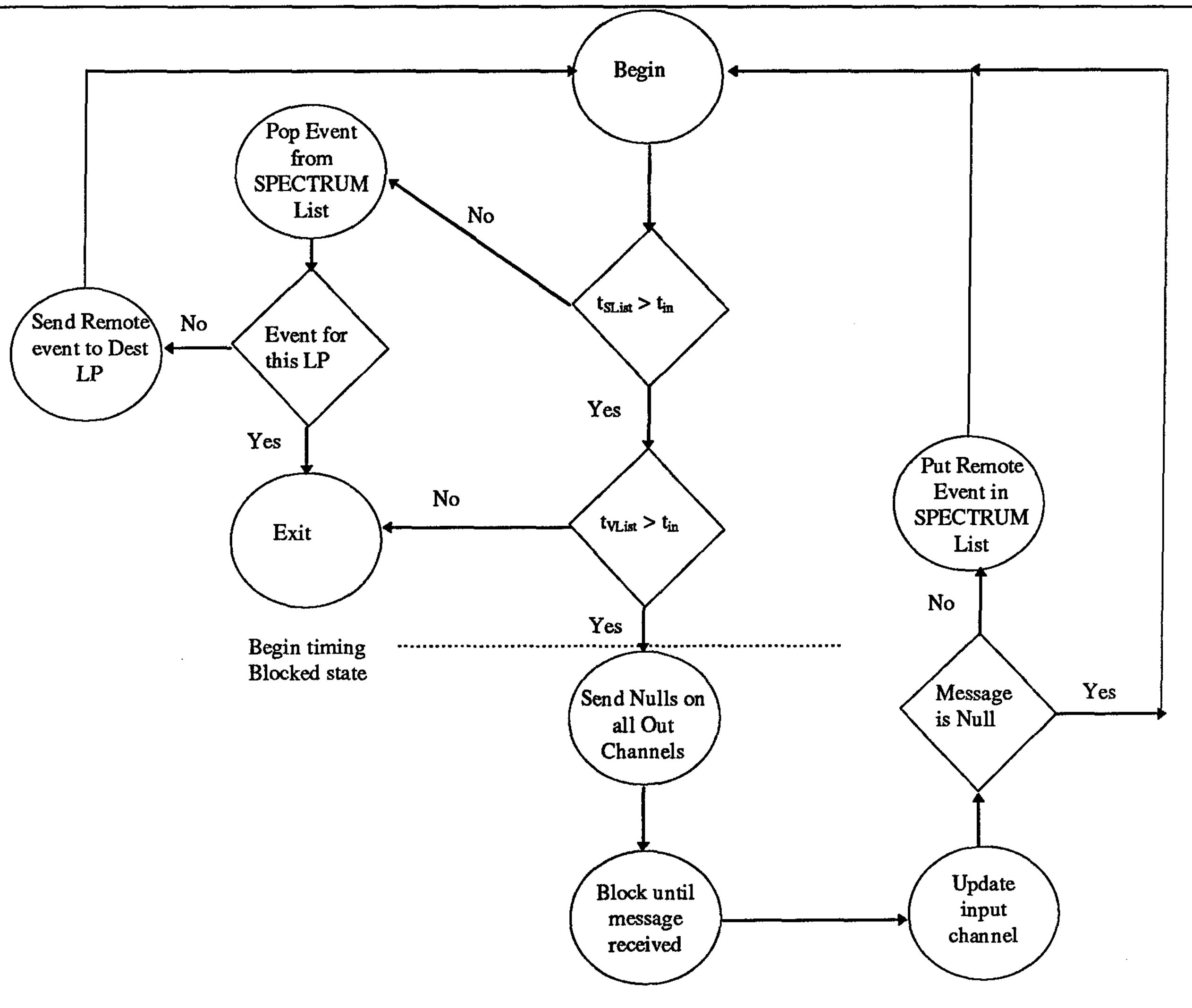
VSIM is the application developed by Comeau and Breeden that simulates VHDL circuits. It is actually an extension of a sequential VHDL simulator by Intermetrics. Consequently, the basic interaction of the application, VSIM, and SPECTRUM is modified from Figure 10. As seen in a VSIM specific reproduction of the previous diagram, Figure 11 shows that VSIM is a simulator unto itself and has its own event list and clock. VSIM, however, has devices to limit its behavior evaluation to some subset of the whole circuit based on the LP number and the LP*x*.map configuration file. Signal changes that affect behaviors mapped to other LPs result in VSIM forming an event with the appropriate destination. VSIM sends the remote event via a call to LP Manager's *Post_Event* function upon retrieving it from the local event list at its designated time. All events are inserted in the VSIM event list. Built around the conservative protocol, VSIM will request an event from SPECTRUM if the time of its next event exceeds the time of the SPECTRUM clock. SPECTRUM must provide a valid event or update the clock for VSIM to continue processing. Thus, VSIM only invokes SPECTRUM to send or receive remote information (events or channel times). Currently, the only access SPECTRUM has to the status of VSIM is *get_low_time()* which provides the time of the earliest event on the VSIM event list.

3.3.1 VHDLClocks 94

VHDLCLOCKS is the null message filter implementation used by Kapp. VHDLClocks94 is a modified version that eliminates some inefficiencies and adds additional instrumentation in order to gain insight into the activities of the processor. For this discussion, let:

t_{in}	: Minimum time of all input channels = t_{safe}
t_{SList}	: Minimum timestamp of event in SPECTRUM Event List
t_{VList}	: Minimum timestamp of event in VSIM Event List
t_{null}	: timestamp on null message = $\mathbf{min}[t_{VList}, t_{in} + \mathit{delay}]$

Figure 12 captures the most complex synchronization of the new protocol. The Get Event filter applies most policies of the protocol in its own code or by invocation of the other filters (not delimited in the figure). The Post Event filter sends nulls down all channels not used by the message being sent and is invoked upon finding a message to another LP in the VSIM event list. The processor can be involved in one of three disjoint states: *application processing, synchronization processing*, and *blocking*. Measuring the time spent in each of these states for all LPs in the simulation may reveal pertinent or even limiting factors of runtime. VHDLClocks94 measures t_{Get} from the invocation of the LP Manager's *Get_Event* routine until it exits, including all time consumed in the Get Event filter. The Get Event filter is the only portion of code that implements blocking.. t_{Block} begins at the demarcation indicated in Figure 12 and ends when the Get Event filter exits (i.e. upon receiving an event that allows application processing to continue). t_{Post} is the time spent in the Post Event Filter. Since these getting and posting events represents almost the entirety of protocol overhead, the sum of t_{Get} and t_{Post} approximates the time spent in synchronization processing and blocking. Since only one LP will be allocated to a processor, task switching overhead can be ignored. Ignoring other operating system induced overheads, the difference between observed runtime for an LP and its measured synchronization overhead is a good estimate



Begin
Pop Event from SPECTRUM List
No
Send Remote event to Dest LP
No
Event for this LP
$t_{SList} > t_{in}$
Yes
Yes
Put Remote Event in SPECTRUM List
No
Exit
$t_{VList} > t_{in}$
No
Yes
Begin timing Blocked state
Send Nulls on all Out Channels
Message is Null
Yes
Block until message received
Update input channel

Figure 12 VHDLClocks94 Get Event

for the application processing time. Summary definitions for an LP are:

t_{Get}	: wall time spent in the LP Manager Get Event subroutine
t_{Post}	: wall time spent in the LP Manager Post Event subroutine
t_{Block}	: wall time spent waiting for a message to allow continued processing
t_{Sync}	: wall time spent processing synchronization protocol routines = $t_{Get} + t_{Post} - t_{Block}$
t_{Proc}	: wall time spent processing the application = $Runtime - (t_{Get} + t_{Post})$

VHDLClocks94 corrects an overflow error. When generating null messages, time stamps may be calculated as the sum of the input channel time and the minimum delay to the specific output channel. Since VSIM operates in picoseconds, a 2000 ns simulation concludes at timestamp 2,000,000,000. The maximum value for the timestamp datatype is 2,147,483,648. Input channels plus delay times were occasionally exceeding this maximum and appearing as negative values. This cause of overflow has been corrected, but the risk is still present in all time calculations when large numbers are used with a restricted datatype.

3.4 Graph Partitioning Tool (GPT)

Kapp developed the Graph Partitioning Tool v2.0 from an earlier version written by Major Eric Christensen, USA. Kapp's most significant improvement from the initial version was to add the ability to perform cost model based partitioning. Structures and procedures were added to allow various measurements of the graph and its allocation. These measurements can then be used as terms in a cost model to ordinally compare two allocations. Kapp's AB Improvement routine uses iterative improvements of the cost model to find better allocations.

Unfortunately, the tool uses a partial cost model to search for iterative improvements. The partial cost model is much more efficient to evaluate, allowing rapid execution of the search. The primary goal of the new version of the Graph Partitioning Tool (version 3.0) is to localize the cost model to a single Ada package and let the AB Annealing routines use the entire cost model in its decisions.

3.4.1 Minimum Delay

Lookahead is ignored in the AB Annealing routines of GPT v2.0. This is not an arbitrary oversight since the calculation of minimum distance for graph partitions is O(*input arcs*$_i$ x n_i^2) where n_i is the number of behaviors in LP_i and *input arcs*$_i$ is the number of input arcs for LP_i. The AB Annealing routine considers a subset of the total number of vertices and must re-evaluate all LPs in the circuit for each allocation raising the cost to O(*numLPs* x *input arcs*$_i$ x n_i^3). With this complexity, it becomes difficult to amortize a week long allocation analysis for a 30 second simulation. While partitioning efficiency is not a goal of this research, reasonable performance is required to progress. As an implementation note, delays are implemented by association with communication channels. Reference to a minimum delay will be equivalently made in terms of the LP pair or the appropriate adjacent channel.

3.4.1.1 Dijkstra

Dijkstra's algorithm is the most efficient (O(n^3)) known for the task of finding the minimum distance between all nodes of a graph with non-negative distances. The actual problem is to find the minimum distance from the subset of input nodes to the subset of output nodes for each LP. Since the graph of a circuit is static, one could find the minimum distances between all nodes and then store this information. Subsequent partitioning analysis could then reference this archived data instead of calculating it at runtime. For a one time cost, efficient use of minimum delay and lookahead can be added to cost model partitioning. Following this reasoning, Dijkstra minimum distance (delay) calculation is offered as a menu option in GPT v3.0.

3.4.1.2 Dynamic Search

Unfortunately, Dijkstra's also carries an O(n^2) space requirement. The 4 Mb of disk space for the Wallace Tree becomes 64 Mb for the 4,243 behavior Associative Memory circuit. Additionally, only knowing the minimum delay of an LP does not allow accurate calculation of Fujimoto's Lookahead Ratio. All paths through an LP contribute to the average delay of that LP. It is the ratio of average delay to minimum delay that defines Lookahead Ratio. There is justification for dynamically measuring all paths through an LP. Therefore, an alternate minimum delay measurement device is offered as a menu option. Dynamic measurement is implemented as a depth first search and may result in unreasonably long partitioning analysis. But, when feasible, it will give better assessment of the delays in an LP.

3.4.1.3 Worst Case Delay

As the circuits get very large, neither of the above options may be reasonable. As a fail-safe option, the user can enter the minimum gate delay to be used as the distance through all LPs. This guarantees the quickest partitioning analysis and correct simulation, but eliminates lookahead as a configurable parameter. Very likely this technique of delay determination will inhibit simulation performance due to the under estimated minimum delay.

3.4.1.4 Infinite Delay

An interesting situation is the occurrence of a source node in one LP and its descendants in adjacent LPs. If no other paths make up such a communication channel, it is not obvious what the delay of that channel should be. Sources have a delay of zero since they perform no processing. However, a channel delay of zero could contribute to a violation of the null protocol if a zero cycle formed between LPs. Based on an understanding of the behavior of the simulation, GPT v3.0 assigns this channel a delay of infinity (implemented as MAX INTEGER). Upon initialization, all events of the testbench are scheduled. Consequently, all the events of a source are in the VSIM event list of the host LP. Since null messages are timestamped **min**[t_{VList}, t_{in} + *delay*], as long as events of the source are in the VSIM event list, t_{VList} will be the timestamp of the null messages down the anomalous channel. The adjacent LP will not receive the infinite delay null message until there are no more events in the VSIM event list. If the VSIM event list empties, an infinite delay may be sent down the source's channel because it will not be generating any more events.

3.4.2 Graph Parameters

The following parameters are measured by the GPT v3.0 statistics gathering package:

Table 3 Graph Partitioning Tool Measured Parameters

Number of Vertices	$Vertices_{Num}$	Count of behaviors in problem graph
Number of Arcs	$Arcs_{Num}$	Count of arcs in problem graph
Number of Components	LPs_{Num}	Number of LPs for which the circuit was allocated
Inter-LP Arcs	$Arcs_{NumInterLP}$	Count of arcs that cross LP boundaries
Channels	$Channels_{Num}$	Count of the channels between LPs; channels encapsulate arcs
Weight of Inter-LP Arcs	$Arcs_{WtInterLP}$	Assuming natural assignment of LPs to processors, a weight w_{ij} can be assigned to communication between any two processors. If a_{ij} is a directed arc that crosses the LP boundary from LP_i to LP_j then $\text{Weight of Inter-LP Arcs} = \sum_i \sum_j a_{ij} \cdot w_{ij}$
Average Weight of Inter-LP Arcs	$Arcs_{AvgWtInterLP}$	Weight of Inter-LP Arcs / Number of Components
Standard Deviation of Output Arcs Weight	$Arcs_{StdWtOut}$	Each LP has a set of output arcs which sums to a weight based on w_{ij} values. Standard deviation is based on the output weight of each LP versus the average output weight for all LPs.
Maximum Deviation of Output Arcs Weight	$Arcs_{MaxWtOut}$	If $OutWeight_j$ is the sum of the weights of output arcs for LP_j, then Maximum Deviation of Output Arcs Weight = $\frac{OutWeight_{Max} - OutWeight_{Avg}}{OutWeight_{Avg}}$
Standard Deviation of Input Arcs Weight	$Arcs_{StdWtIn}$	Similar to output arc counterpart but for arcs that cross the LP boundary into LPs. This attempts to assess the balance of communication demands for both input and output messages.
Maximum Deviation of Input Arcs Weight	$Arcs_{MaxWtIn}$	Similar to output arc counterpart. Since runtime is based on max runtime of the LPs, it makes sense to track max parameters as well as average and standard deviation.
Average Lookahead Ratio	$Lookahead_{Avg}$	Fujimoto defines Lookahead Ratio as average message delay over minimum message delay. Only path lengths through the LP are available a priori to estimate average delays. Actual average with depend on event frequency down the various paths as well as event generation and termination within the LP. Note that LPs will have different delays for different pairs of

		input and output nodes. The overall minimum does not apply to output nodes not on that path. Thus, the minimum distances for all input-output node pairs in an LP is calculated resulting in a single minimum path measure for each output channel. Lookahead Ratio is defined for a single LP. There is no aggregate measure for the whole simulation. Consequently, the aggregate measure used here is found by summing the average paths in all LPs and dividing by the sum of minimum paths through all LPs: $$Average\ Lookahead = \frac{\frac{\sum_{All\, i \neq j} delay_{ij}}{\#\, of\ paths\ from\ LP_i\ to\ LP_j}}{\sum_{All\, i \neq j} \min(delay_{ij})}$$ Note that the different techniques for finding minimum delay will produce different values of Average Lookahead since they track different numbers of total paths.
Standard Deviation of Lookahead	$Lookahead_{Std}$	Lookahead Ratio is calculated for each LP and compared to the average over all LPs.
Maximum Deviation of Lookahead	$Lookahead_{Max}$	Same calculation technique as other maximum quantities.
Average Computational Load	$Load_{Avg}$	Each behavior b_i is assigned a computational weight cw_i. Average load for all LPs is: $$\frac{\sum b_i \cdot cw_i}{Number\ of\ LPs}$$
Maximum Deviation Computational Load	$Load_{Max}$	Same calculation technique as other maximum quantities.
Standard Deviation Computational Load	$Load_{Std}$	Each LP load compared to the overall average.
Time spent in Application Processing	t_{Proc}	Sum of time spent by each LP in application processing for an allocation
Time spent in Synchronization Processing	t_{Sync}	Sum of time spent by each LP in synchronization processing for an allocation
Time spent in Blocking	t_{Block}	Sum of time spent by each LP in blocking for an allocation
Maximum Runtime	t_{MaxRun}	Runtime taken by last LP to complete processing.
Speedup	*Speedup*	Time for biased sequential execution / time for parallel allocation execution
Nulls Sent	$Nulls_{Num}$	Total number of nulls actually sent by all LPs during a simulation
VList Nulls Posed	$Nulls_{VList}$	Total number of nulls posed by Get Event filter with the timestamp of the earliest event of the VSIM event list
Delay Nulls Posed	$Nulls_{Delay}$	Total number of nulls posed by Get Event filter with the

		timestamp of the earliest input channel plus the minimum delay
Total Nulls Posed	$Nulls_{Posed}$	Sum of $Nulls_{VList}$ and $Nulls_{Delay}$. Incorrectly ignores nulls sent as part of Post Event filter. $Nulls_{Posed}$ will exceed $Nulls_{Num}$ since some nulls posed will not increase the channel time so VHDLClocks94 will not send them.
Total Reals Sent	$Reals_{Num}$	Count of real messages sent by all LPs in a simulation.

3.5 Wallace Tree

The majority of data collection and model formulation is based on the same Wallace Tree Multiplier used by Breeden and Kapp. This circuit is large enough to permit conclusions while small enough to wield in an experimental environment. The circuit consists of 1,050 behaviors and 1,770 arcs. It accepts two 8 bit input vectors to form their 12 bit product.

3.6 Associative Memory

The associative memory circuit developed by Kapp consists of 4,234 behaviors and 9,312 arcs (Kapp, 1993:89). It is the largest circuit simulated at AFIT, requiring 20-40 minutes to simulate on the iPSC/2. Implementing a 16x16 memory array, the testbench performs several read and write tests including pattern searches. Kapp reports that design conveniences may hinder simulation performance. A common enable signal causes all registers to schedule events, although only one may be of interest in a particular cycle (e.g. read versus write registers).

3.7 Model Structure

The cost model proposed in this research was developed by collecting several hundred samples of Wallace Tree statistics for the various partitioning strategies on 2 through 8 processors. By using the AB Improvement routine a representative spectrum of values for all parameters was pursued. Blind use of three different cost models, allowed data to be gathered while avoiding bias to a particular region of the search space.

Runtime performance is a function of many disparate parameters and their interrelationships. In order to add theoretic structure to the model development, an abstract mathematical description is now developed. Since processors are in one of three states at all times, the sum of time spent for each state should give the runtime of that processor. While the critical runtime is the maximum over all LPs, the sum of all states over all LPs is guaranteed to include the performance of the slowest. Also, if LPs are generally balanced, the sum over all LPs will be a good indicator for the performance of the slowest LP. Rather than attempt to build a cost model for runtime directly, cost models were built for the component states: *application processing*, *synchronization processing*, and *blocking*. Upon determining the component cost models, the terms of each were used to construct the overall model. These subordinate cost models provide accuracy and reliability data for each component, allowing a more insightful critique of the composite cost model. All cost models were developed in accordance with the procedure outlined in Section 2.7.

$$t_{runtime} = b_0 + b_1 \cdot \sum_{AllLPs} t_{Proc} + b_2 \cdot \sum_{AllLPs} t_{Sync} + b_3 \cdot \sum_{AllLPs} t_{Block}$$

Equation 8 Top Level Cost Model

3.8 Summary

This chapter outlined the operating environment and specific test cases used in this research. The three major software systems supporting this project are: SPECTRUM, VSIM, and the Graph Partitioning Tool v3.0. The specific software used for synchronization management is the null message filter VHDLClocks94 which is instrumented to measure the time spent by the processor in each of three states: application processing, synchronization processing, and blocking. Only a summary description of the support software behavior is presented here. Each system has several thousand lines of code and demands an in depth study to be fully understood.

By predicting the time spent in the three component states, a cost model is formed to predict overall runtime. The statistical methods used exactly follow those presented by Thorndike and are outlined in Section 2.7 of this paper. The analysis is primarily performed on the Wallace Tree Multiplier of previous AFIT research. The cost model is then applied to Kapp's Associative Memory circuit to assess robustness.

4. *Data Description and Analysis*

4.1 Model Development

This section presents summary forms of the sample data collected and the development of the cost model. Three hundred fifty eight simulations for the Wallace Tree Multiplier comprised the entire sample set. Each data tuple was randomly assigned into one of two sets, Set A or Set B. All cost models were developed from each dataset independently and then compared in accordance with double cross-validation. Upon validation of the terms, each cost model was redeveloped over the entire sample set. To avoid ambiguity between the various cost models under development, the following names apply to this discussion:

$Cost_{Proc}$	: predicts total time of all LPs spent in application processing
$Cost_{Sync}$	: predicts total time of all LPs spent in synchronization processing
$Cost_{Block}$	: predicts total time of all LPs spent in blocking
$Cost_{Runtime}$	: the ultimate model of this research, predicts the maximum LP runtime built from the assumed model that

$$Cost_{Runtime} = b_0+b_1Cost_{Proc}+b_2Cost_{Sync}+b_3Cost_{Block}$$

Equation 9 $\mathbf{Cost_{Runtime}}$ Top Level Model

All cost models use *a priori* parameter values for a specific allocation as input.

4.1.1 Top Level Model

Before forming cost models for the component terms, the top level model should be validated. A linear regression of Equation 8 over the entire sample population results in the following coefficient vector

$$\beta = \begin{bmatrix} -2.6688 \\ 0.81 \\ 0.091 \\ 0.0907 \end{bmatrix}.$$

Thus, measured t_{Proc}, t_{Sync}, and t_{Block} can be used to form a $Cost_{Runtime}$ linear model with 0.978 correlation and average error of prediction of 1.006 seconds. The correlation seems solid and the absolute error measure is within 4% of the average runtime. Therefore, if *a priori* parameters can be found to predict t_{Proc}, t_{Sync}, and t_{Block}, the component cost models can be accumulated to predict t_{MaxRun}.

4.1.2 Application Processing Model

The initial step of model development is identifying a candidate set of terms (parameters) pertinent to the criterion term which is to be predicted. An initial set is formed from the parameters that demonstrate significant correlation to the criterion. Appendix B presents multiple correlation data for all parameters in the sample set. From this, $Arcs_{NumInterLP}$, $Arcs_{StdWtOut}$, $Arcs_{StdWtIn}$, and $Lookahead_{Avg}$ are added to the candidate set. The $Cost_{Proc}$ model offers the greatest potential for prediction accuracy since t_{Proc} is defined to be independent of the complex synchronization protocol. It is logical that load and load balance will influence the time spent in application processing. Consequently, $Load_{Max}$, $Load_{Avg}$, $Load_{Std}$ are added to the candidate set. An LP's application processing time should be fundamentally dependent on the number of events it has to process. A crude complexity analysis can be accomplished by calling E the set of all events that will be processed in a simulation. Note that E is constant over all allocations. In order to accomplish E, VSIM must enqueue and process all events over the course of the simulation. The implementation of the next event queue therefore has direct relevence to the performance of the simulation. Insertion into the next event queue sort is O(N) in the worst case where $N = |E|$. Consequently, the entire simulation performs O(N^2) of application processing. Assuming uniform gate activity and balanced loads on each partition, the amount of application processing for each LP should be O $((\frac{N}{P})^2)$. Note that this implies that application processing can achieve superlinear speedup since O$((\frac{N}{P})^2)$ is less than O(N^2/p). However, this was an extremely crude analysis which should not be taken too literally. Its purpose was to identify additional terms to add to the candidate set. Combinations of the *1/p* term are included to complete the candidate set of terms for $Cost_{Proc}$. The candidate set is:

$$\{ Arcs_{StdWtOut}, Arcs_{StdWtIn}, Lookahead_{Avg}, Load_{Max}, Load_{Std}, 1/[LPs_{Num}^2\log(LPs_{Num})], \\ 1/LPs_{Num}^2, \log(LPs_{Num}), 1/LPs_{Num}\}.$$

Note that $Load_{Avg}$ was dropped from the candidate set. Since the $Vertices_{Num}$ is constant between allocations, $Vertices_{Num}/LPs_{Num}$ offers no more information than just $1/LPs_{Num}$. The corresponding coefficient of the linear model will incorporate the $Vertices_{Num}$ constant.

The mathematical software used for cost model development is MathCad 5.0+ for Microsoft Windows. Datasets are arrays where each row is a single sample instance and each column is a specific parameter. A (row,column) pair identifies the value of a single parameter for a single simulation run. For example, $SetA_{20,LoadMax}$ is the $Load_{Max}$ value for the twentieth sample of Set A. Since not all parameters are considered during model construction, a subset array X is the data array of the candidate set. Let x be the array including a specific subset of candidates. Let y be the sample vector of the criterion parameter. The development of $Cost_{Proc}$ follows:

Initialize candidate data array:

$i := 0..\text{rows}(\text{SetA}) - 1 \qquad \text{lg}(x) := \dfrac{\log(x)}{\log(2)}$

$X_{i,0} := 1$

$X_{i,1} := \left(\dfrac{1}{\text{SetA}_{i,\text{LPsNum}}}\right)^2 \qquad X_{i,2} := \left(\dfrac{1}{\text{SetA}_{i,\text{LPsNum}}}\right) \qquad X_{i,3} := \text{SetA}_{i,\text{LoadStd}} \qquad X_{i,4} := \text{SetA}_{i,\text{LoadMax}}$

$X_{i,5} := \text{lg}\left(\dfrac{1}{\text{SetA}_{i,\text{LPsNum}}}\right) \qquad X_{i,6} := \text{SetA}_{i,\text{ArcsWtInterLP}} \qquad X_{i,7} := \text{SetA}_{i,\text{ArcsStdWtIn}} \qquad X_{i,8} := \text{SetA}_{i,\text{ArcsStdWtOut}}$

$X_{i,9} := \text{SetA}_{i,\text{LookaheadAvg}} \qquad X_{i,10} := X_{i,1} \cdot X_{i,5} \qquad y_i := \text{SetA}_{i,\text{tProc}}$

Each row of *X* has candidate parameter values for a single simulation. Each column of *X* has the values over all SetA runs of a specific candidate parameter. The first column of *X* is initialized to a vector of 1s to keep β and *X* of compatible dimensions.

Choose the first parameter to add to the model:

$r^T =$

	0	1	2	3	4	5	6	7	8	9	10
0	0	0.972	0.98	-0.444	-0.465	0.951	-0.211	0.586	0.383	0.518	-0.973

Vector *r* is the multiple correlation vector for each *X* column and the *y* vector. The second column of *X* carries the greatest correlation to the criterion parameter so the initial linear regression equation is

$$Cost_{Proc} = \beta_0 + \beta_1 \cdot X_2 \quad or \quad Cost_{Proc} = \beta_0 + \beta_1 \cdot \frac{1}{SetA_{LPsNum}}$$

Matrix *x* is used to collect the selected columns of *X* into a single matrix.

$x_{i,0} := 1 \qquad x_{i,1} := X_{i,2}$

$\beta := \left(x^T \cdot x\right)^{-1} \cdot x^T \cdot y \qquad \beta = \begin{pmatrix} 12.312 \\ 50.264 \end{pmatrix}$

$p := \mathrm{cols}(x) \qquad p = 2$
$k := p - 1 \qquad k = 1$
$n := \mathrm{rows}(x) \qquad n = 179$

$$SSr := \left[\beta^T \cdot x^T \cdot y - \frac{\left(\sum_{i=0}^{n-1} y_i \right)^2}{n} \right]_{0,0} \qquad SSe := y^T \cdot y - \beta^T \cdot x^T \cdot y \qquad MSe := \left(\frac{SSe}{n-k-1} \right)_{0,0}$$

$$F := \frac{\frac{SSr}{k}}{MSe} \qquad \mathrm{Fstat} = F_{\alpha,k,n\text{-}p} = 2.71$$

$F = 4.196 \cdot 10^3$

$\sqrt{MSe} = 1.313$

$SSr02 := SSr$

The F statistic is used to verify that the model is not just coincidentally related to the criterion. Using an α of 0.1, the calculated *F* must be greater than *Fstat* to reject the null hypothesis. The large number of samples encourages that any relationship observed is likely to actually exist. The MSe is the mean squared residual error. Thus, on average, the model will predict $\sqrt{MSe}$ from observed criterion values due to error not accounted for in the regression. *SSr02* saves the amount of error accounted for in the regression for later marginal comparison with a larger model. To select the next parameter to add to the model, each remaining candidate will be included with the current model, individually.

Add additional terms to the model:

$x := 0$

$x_{i,0} := 1 \qquad x_{i,1} := X_{i,2} \quad x_{i,2} := X_{i,6}$

$\beta := \left(x^T \cdot x\right)^{-1} \cdot x^T \cdot y \qquad\qquad \beta = \begin{pmatrix} 9.954 \\ 53.595 \\ 0.003 \end{pmatrix}$

$p := \text{cols}(x) \quad p = 3$
$k := p - 1 \quad k = 2$
$n := \text{rows}(x) \quad n = 179$

$$SSr := \left[\beta^T \cdot x^T \cdot y - \frac{\left(\sum_{i=0}^{n-1} y_i \right)^2}{n} \right]_{0,0} \qquad SSe := y^T \cdot y - \beta^T \cdot x^T \cdot y \qquad MSe := \left(\frac{SSe}{n - k - 1} \right)_{0,0}$$

$$F := \frac{\frac{SSr}{k}}{MSe} \qquad \text{Fstat} = F_{\alpha,k,n-p} = 2.30$$

$F = 6.23 \cdot 10^3$

$\sqrt{MSe} = 0.773$

$SSr026 := SSr$

In the above calculations, testing each remaining candidate as another term in the model resulted in the F statistics in Table 4. Note that X_2 was not considered for addition again. Since X_6 resulted in the highest F statistic, it is selected for addition to the model. The addition of a new parameter to the model raises the question of the marginal contribution of that parameter in lieu of those already present. Additionally, does the addition of this new term negate the necessity for other terms already in the model? Using the partial F test of Section 2.7, each term is removed from the model. The marginal decrease in SSr is tested for significance as follows:

Candidate Parameter	Resulting F statistic
X_1	2188
X_3	2463
X_4	2486
X_5	2213
X_6	*6230*
X_7	2395
X_8	2772
X_9	3771
X_{10}	2143

Table 4 F Statistic Comparison for Parameter Selection

$x := 0$

$x_{i,0} := 1 \qquad x_{i,1} := X_{i,6}$

$\beta := \left(x^T \cdot x\right)^{-1} \cdot x^T \cdot y \qquad \beta = \begin{pmatrix} 26.762 \\ -0.004 \end{pmatrix}$

$p := \mathrm{cols}(x) \quad p = 2$
$k := p - 1 \quad k = 1$
$n := \mathrm{rows}(x) \quad n = 179$

$$SSr := \left[\beta^T \cdot x^T \cdot y - \frac{\left(\sum_{i=0}^{n-1} y_i\right)^2}{n}\right]_{0,0} \qquad SSe := y^T \cdot y - \beta^T \cdot x^T \cdot y \qquad MSe := \left(\frac{SSe}{n-k-1}\right)_{0,0}$$

$Fp := \dfrac{SSr026 - SSr}{MSe} \qquad Fstat = F_{\alpha,k,n-p} = 2.71$

$Fp = 174.419$

As seen in the *Fp* statistic, the *SSr* of the model with X_2 and X_6 is significantly more than just using X_6. Therefore, both terms should stay.

The model continues to be developed by using the F statistic to select from the remaining candidate set and then test the significance of each term in the model. Soon, additional terms do little to improve the accuracy of the model upon which the process is concluded. For the SetA development of $Cost_{Proc}$, the model is:

$$Cost_{Proc} = \beta_0 + \beta_1 \cdot (\frac{1}{LPs_{Num}}) + \beta_2 \cdot (Arcs_{NumInterLP}) + \beta_3 \cdot (Arcs_{StdWtIn}) + \beta_4 \cdot (\frac{1}{LPs_{Num}{}^2}) + \beta_5 \cdot (Arcs_{StdWtOut})$$

Equation 10 ***$Cost_{Proc}$***

$$\beta = \begin{bmatrix} 10.185 \\ 37.734 \\ 0.004 \\ 0.041 \\ 21.456 \\ 0.005 \end{bmatrix} \blacksquare$$

$$\text{corr}(\text{CostProc}, y) = 0.995 \quad \sqrt{\text{MSe}} = 0.633 \blacksquare$$

As can be seen, the $Cost_{Proc}$ for SetA is very successful in both correlation and absolute error. The identical form of the model was developed over SetB with the following coefficients and summary statistics:

$$\beta = \begin{bmatrix} 10.135 \\ 39.92 \\ 0.004 \\ 0.039 \\ 15.203 \\ 0.011 \end{bmatrix} \blacksquare$$

$$\text{corr}(\text{CostProc}, y) = 0.995 \blacksquare \sqrt{\text{MSe}} = 0.656 \blacksquare$$

Double cross-validation requires that the β for SetA be used for SetB and visa versa. Upon performing this,

	SetA	SetB
βa	Corr = 0.995 $\sqrt{MSe} = 0.633$	Corr = 0.995 $\sqrt{MSe} = 1.895$
βb	Corr = 0.995 $\sqrt{MSe} = 1.632$	Corr = 0.995 $\sqrt{MSe} = 0.656$

Although the absolute error increased in cross-validation, both the form and the correlation of the model applied well. The model is validated. Regressing this model on the entire sample set yields:

$$\beta = \begin{bmatrix} 10.109 \\ 38.75 \\ 0.004 \\ 0.041 \\ 18.267 \\ 0.008 \end{bmatrix}$$

$$\mathrm{corr}(\mathrm{CostProc}, y) = 0.995 \qquad \sqrt{\mathrm{MSe}} = 0.651$$

4.1.3 Synchronization Processing Model

Synchronization costs present a much more formidable challenge in identifying candidate parameters. Correlations are weaker and fewer than with t_{Proc}. Since t_{Sync} is a function of many parameters, it becomes difficult to interpret two dimensional projections of the *k*-space within which the t_{Sync} criterion is varying. Figure 13 presents a repeating projection pattern for both t_{Sync} and t_{Block}. Although the relationship is essentially linear, two disparate lines demonstrate the influence of other parameters. Some other parameter is discretely (not continuously) affecting the slope of the t_{Sync} versus $parameter_x$ relationship. By sorting the sample set on the independent variable ($parameter_x$) and then observing consistent changes in other parameters of the dataset, the terms varying with slope can be identified. In the case of LPs_{Num} the companion parameter is $Arcs_{NumInterLP}$,(See Figure 14). By using this technique, the final form of the $Cost_{Sync}$ model performs very well over the sample set.

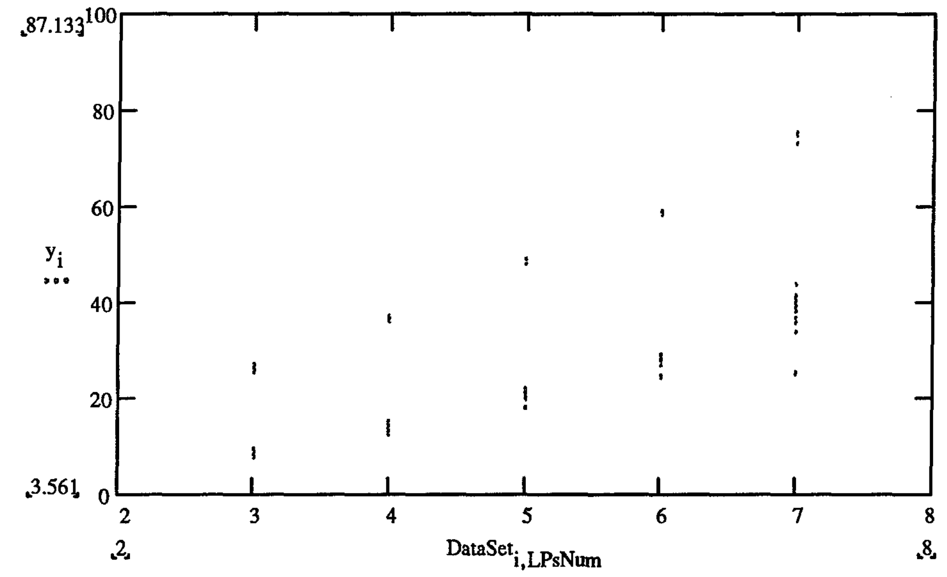


Figure 13 t_{Sync} **versus** LPs_{Num}

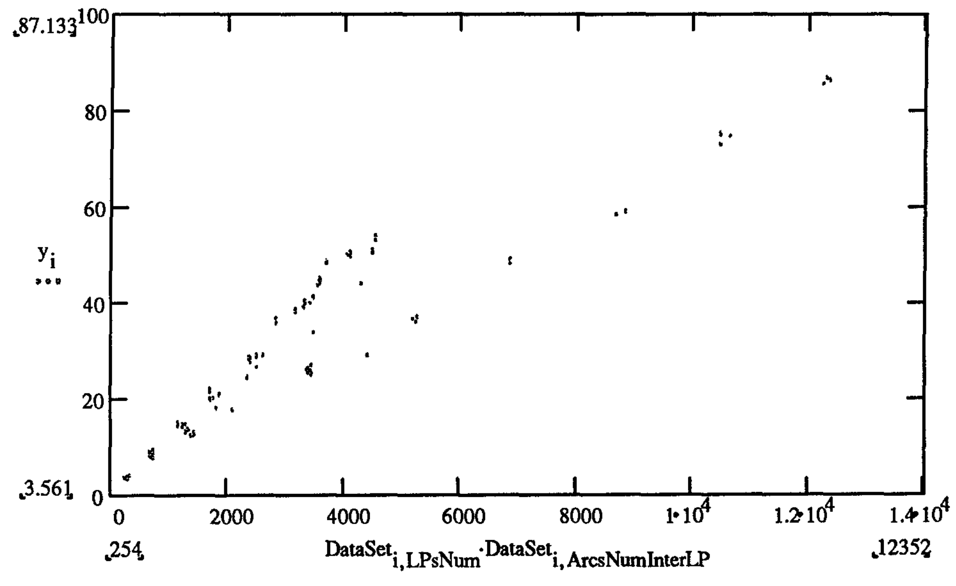


Figure 14 t_{Sync} **versus** LPs_{Num} **x** $Arcs_{NumInterLP}$

$$Cost_{Sync} = \beta_0 + \beta_1 \cdot (LPs_{Num} \cdot Arcs_{NumInterLP}) + \beta_2 \cdot (LPs_{Num}{}^2) + \beta_3 \cdot (Arcs_{StdWtOut}) + \beta_4 \cdot (Lookahead_{Avg}) + \beta_5 \cdot (LPs_{Num})$$

Equation 11 *$Cost_{Sync}$*

For SetA

$$\beta = \begin{bmatrix} 16.386 \\ 0.005 \\ 0.453 \\ -0.094 \\ -0.51 \\ -1.77 \end{bmatrix}.$$

For SetB

$$\beta = \begin{bmatrix} 12.001 \\ 0.006 \\ 0.489 \\ -0.06 \\ -0.285 \\ -1.801 \end{bmatrix}.$$

Cross-validation

	SetA	SetB
βa	Corr = 0.994 $\sqrt{MSe} = 2.107$	Corr = 0.987 $\sqrt{MSe} = 7.657$
βb	Corr = 0.993 $\sqrt{MSe} = 7.773$i	Corr = 0.99 $\sqrt{MSe} = 2.305$

While not as accurate as *$Cost_{Proc}$*, *$Cost_{Sync}$* shows good correlation and absolute error within 9% of the average t_{Sync}. Cross-validation kept good correlation, but absolute error jumped to 30% of the average t_{Sync}. This suggests that the ordinal scaling of $Cost_{Sync}$ is correct, but its ability to retain interval information is poor.

For the whole sample set:

$$\beta = \begin{bmatrix} 13.787 \\ 0.005 \\ 0.462 \\ -0.078 \\ -0.38 \\ -1.639 \end{bmatrix}$$

$$\mathrm{corr}(\mathrm{CostSync}, y) = 0.992 \qquad \sqrt{\mathrm{MSe}} = 2.244$$

4.1.4 Blocking Model

Modeling blocking proves to be the most difficult of the three components to predict. While similar difficulties as with $Cost_{Sync}$ are encountered, where the $Cost_{Sync}$ issues are resolved, $Cost_{Block}$ remains a problem. The combining technique of the previous section failed to collapse the disparate linear relationships into one. Additionally, t_{Block} varies over a much larger range resulting in a similarly large discrepancy between variance due to regression terms and total variance. Additionally, the models formed on SetA and SetB failed to include the same terms. Without insight into alternative parameter combinations, both models were cross validated and the best chosen for regression over the whole sample set. Note, the chosen model failed double cross-validation, but is the best one available.

$$Cost_{Block} = \beta_0 + \beta_1 \cdot (LPs_{Num}{}^2) + \beta_2 \cdot (LPs_{Num} \cdot Arcs_{NumInterLP}) + \beta_3 \cdot (Lookahead_{Avg}) + \beta_4 \cdot (e^{\frac{1}{LoadAvg}}) + \beta_5 \cdot (LPs_{Num})$$

Equation 12 $Cost_{Block}$

For SetA

$$\beta = \begin{bmatrix} 4.758 \cdot 10^{6} \\ 5.503 \\ 0.007 \\ -2.65 \\ -4.758 \cdot 10^{6} \\ 4.52 \cdot 10^{3} \end{bmatrix}$$

For SetB

$$\beta = \begin{bmatrix} 2.873 \cdot 10^{6} \\ 4.629 \\ 0.002 \\ -1.294 \\ -2.873 \cdot 10^{6} \\ 2.731 \cdot 10^{3} \end{bmatrix}$$

Cross-validation

	SetA	SetB
βa	Corr = 0.987 $\sqrt{MSe}$= 11.186	Corr = 0.978 $\sqrt{MSe}$= 75.604
βb	Corr = 0.984 $\sqrt{MSe}$= 53.353	Corr = 0.98 $\sqrt{MSe}$= 12.767

For whole sample set

$$\beta = \begin{bmatrix} 4.307 \cdot 10^{6} \\ 5.257 \\ 0.005 \\ -2.107 \\ -4.307 \cdot 10^{6} \\ 4.093 \cdot 10^{3} \end{bmatrix}$$

$$\text{corr(CostBlock, y)} = 0.983 \qquad \sqrt{\text{MSe}} = 12.229$$

While the correlation remains impressive, the average error of prediction is now 15% of the average t_{Block}. In cross-validation that error jumps to over 90%. This model is seemingly unreliable with regard to interval scaling. Hopefully, that phenomenon will dissolve in the composite model for runtime.

4.1.5 Composite Model

The composite model development for $Cost_{Runtime}$ places all the terms of the component models into the candidate set. The candidate matrix is initialized by:

$$X_{i,0} := 1$$

$$X_{i,1} := DataSet_{i,LPsNum} \qquad X_{i,2} := \left(DataSet_{i,LPsNum}\right)^2 \qquad X_{i,3} := \left(\frac{1}{DataSet_{i,LPsNum}}\right)^2$$

$$X_{i,4} := DataSet_{i,ArcsStdWtOut} \qquad X_{i,5} := (e)^{\frac{1}{DataSet_{i,LoadAvg}}} \qquad X_{i,6} := DataSet_{i,LookaheadAvg}$$

$$X_{i,7} := \left(\frac{1}{DataSet_{i,LPsNum}}\right) \qquad X_{i,8} := DataSet_{i,LPsNum} \cdot DataSet_{i,ArcsNumInterLP} \qquad X_{i,9} := DataSet_{i,ArcsWtInterLP}$$

$$X_{i,10} := DataSet_{i,ArcsStdWtIn}$$

The correlation vector of these parameters with observed t_{MaxRun} is

$$r^T = \begin{array}{c|ccccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \hline 0 & 0 & 0.507 & 0.607 & -0.088 & -0.446 & 0.508 & 0.081 & -0.217 & 0.67 & 0.475 & -0.375 \end{array}.$$

As can be seen, the correlations are not nearly as strong as those observed for $Cost_{Proc}$. The model development, however, is surprisingly well-behaved. Both SetA and SetB select identical terms for the model.

$$Cost_{Runtime} = \beta_0 + \beta_1 \cdot (LPs_{Num}{}^2) + \beta_2 \cdot (\frac{1}{LPs_{Num}}) + \beta_3 \cdot (LPs_{Num} \cdot Arcs_{NumInterLP}) + \beta_4 \cdot (Lookahead_{Avg}) + \beta_5 \cdot (Arcs_{StdWtOut})$$

Equation 13 $Cost_{Runtime}$

For SetA

$$\beta = \begin{bmatrix} 7.908 \\ 0.251 \\ 52.273 \\ 0.002 \\ -0.39 \\ -0.013 \end{bmatrix}$$

For SetB

$$\beta = \begin{bmatrix} 7.701 \\ 0.264 \\ 47.171 \\ 0.001 \\ -0.236 \\ -0.002 \end{bmatrix}$$

Cross-validation

	SetA	SetB
βa	Corr = 0.935 $\sqrt{MSe} = 1.88$	Corr = 0.868 $\sqrt{MSe} = 2.962$
βb	Corr = 0.919 $\sqrt{MSe} = 5.926$	Corr = 0.872 $\sqrt{MSe} = 2.195$

For whole sample set

$$\beta = \begin{bmatrix} 7.908 \\ 0.251 \\ 52.273 \\ 0.002 \\ -0.39 \\ -0.013 \end{bmatrix}$$

$\mathrm{corr}(\mathrm{CostRuntime}, y) = 0.935 \quad \sqrt{\mathrm{MSe}} = 1.88$

4.1.6 Minimum Delay Revisited

Unfortunately, the $Cost_{Runtime}$ model raises an issue. The minimum delay has not been resolved for very large circuits like the Associative Memory. Dijkstra's would require a 64 Mb data structure on disk while the dynamic search performs too slowly for iterative improvement of a 4,000 node circuit. It would have been convenient if the cost model allowed ignoring Lookahead which would prevent the need to find the minimum delay. Unfortunately, the issue must be forced by redeveloping $Cost_{Runtime}$ without $Lookahead_{Avg}$ as a candidate. Let this cost model be labeled $Cost_{xRuntime}$ and it is

$$Cost_{xRuntime} = \beta_0 + \beta_1 \cdot (LPs_{Num}) + \beta_2 \cdot (\frac{1}{LPs_{Num}{}^2}) + \beta_3 \cdot (LPs_{Num} \cdot Arcs_{NumInterLP})$$

Equation 14 ***$Cost_{xRuntime}$***

for the whole sample set

$$\beta = \begin{bmatrix} 5.852 \\ 0.298 \\ 44.196 \\ 0.001 \end{bmatrix}$$

$$\text{corr}(\text{CostxRuntime}, y) = 0.894 \quad \sqrt{\text{MSe}} = 2.176$$

While the correlation has diminished from the component cost models, it still appears well correlated and average error is less than 8% of the average runtime. Except for the limited degrees of freedom, there is no reason to doubt this model. Note that the only pertinent parameters are manipulations of LPs_{Num} and $Arcs_{NumInterLP}$. There is a linear term that reflects the increase in communication as LPs_{Num} increases as well as a inverted square term that decreases application processing time as LPs_{Num} increases. The few unique parameters legitimize their importance in accelerating VHDL simulations, but imply the model is making assumptions regarding other parameters. $Load_{Avg}$ is an obvious influence to runtime; however, when developing over a single circuit, the term is redundant to $1/LPs_{Num}$ since $Vertices_{Num}$ is constant. Since no load balancing term remained in the cost model, it is likely the sample set failed to expose enough variance in $Load_{Std}$ to introduce it into the model. The following section presents speedup curves for the various types of partitioning.

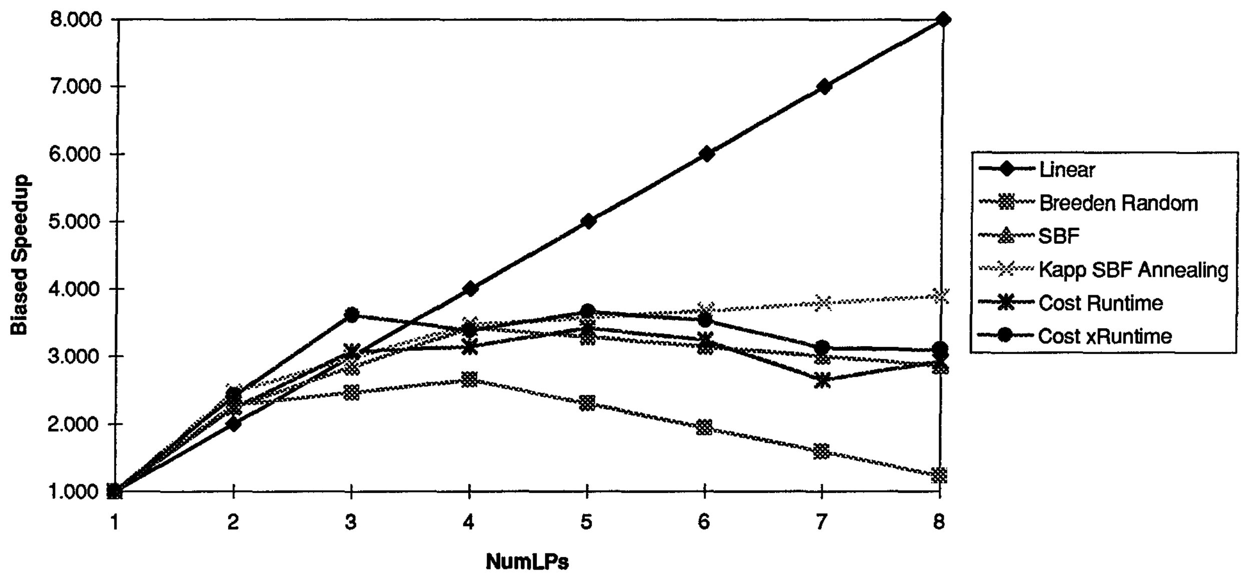


Figure 15 Speedup Curves with Cost Models

4.2 Model Verification

Two hundred eight additional runs comprise the validation set. All cost models were evaluated on their ability to improve initial partitions via AB Improvement. Figure 15 shows the results of the AB Annealing routine using the $Cost_{Runtime}$ and $Cost_{xRuntime}$ models for simulation of the Wallace Tree Multiplier. Performance is disappointing and difficult to interpret. Foremost,

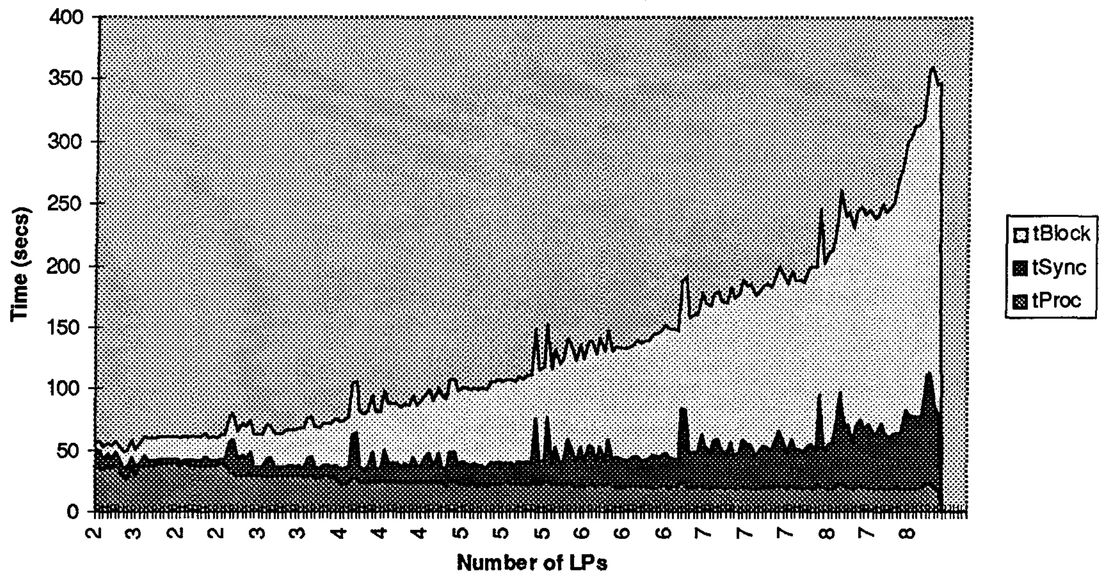


Figure 16 Sum of LP Activity versus Number of LPs

both statistical models show conflicting results in contrast with the Kapp model. A margin is given to the Kapp model in that instrumentation of the three processing states adds about a 10% performance overhead. This overhead would not be realized in sequential execution since the filter is not called and consequently diminishes speedup. However, both models developed in this thesis demonstrate erratic speedup patterns with isolated points of extremely good or extremely bad performance relative to the preceding curve. Inspection of the data reveals that continued iteration using either cost model may result in performance degradation. $Cost_{Runtime}$ frequently lessens speedup over the SBF partition on which it was based.

The next step involves identifying the sources of error for the cost models. Figure 16 demonstrates the behavior of the validation set as the number of LPs increases. Note that cumulative processing time decreases. In other words, partitioning the simulation requires less total work to be accomplished and provides the opportunity for superlinear speedup. Of course, partitioning the sequential implementation would also realize this reduction in overall work.

Blocking seems to dominate LP performance beginning with 4 LP partitioning. Next, each component cost model is compared to appropriate values of the validation set. $Cost_{Proc}$ maintains a correlation of 0.993 with t_{Proc} (see Figure 17 $Cost_{Proc}$ versus t_{Proc}). $Cost_{Sync}$ and $Cost_{Block}$ report equally impressive correlation with their observed counterparts. Correlation for $Cost_{Sync}$ and t_{Sync} is 0.951 and correlation for $Cost_{Block}$ and t_{Block} is 0.944. Despite this prowess at predicting time spent in the component states, $Cost_{Runtime}$ and $Cost_{xRuntime}$ deliver correlations of 0.722 and 0.727 respectively. Somehow the sum of the parts is not equaling the whole. An interesting phenomenon occurs when comparing the top-level component model (Equation 8) using observed components and then calculated components (Equation 9). When using observed data of the validation set Equation 8 exhibits a correlation of 0.954 with t_{MaxRun}. However, when using the component cost models, Equation 9 demonstrates a correlation of 0.765 to t_{MaxRun}. Correlation ($\succ$) fails to survive through the linear model! Put in symbolic terms:

$$\begin{array}{l} a \succ a' \quad b \succ b' \quad c \succ c' \\ f(a',b',c') \succ Y \text{ but} \\ f(a,b,c) \not\succ Y \end{array}$$

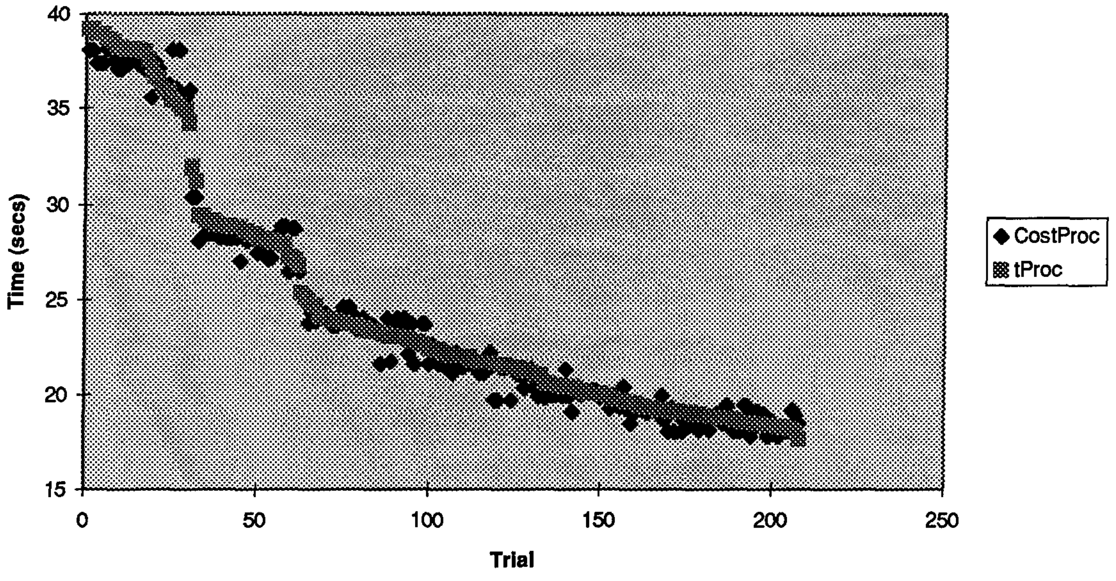


Figure 17 $Cost_{Proc}$ versus t_{Proc}

Of course, the final cost model does not include the component cost models, only the terms

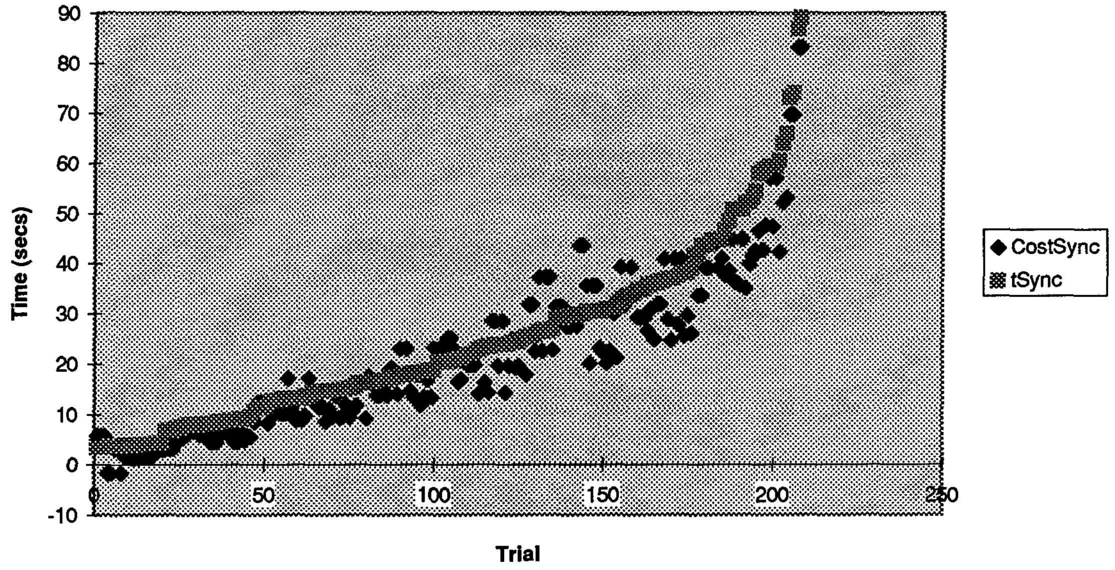


Figure 18 $Cost_{Sync}$ versus t_{Sync}

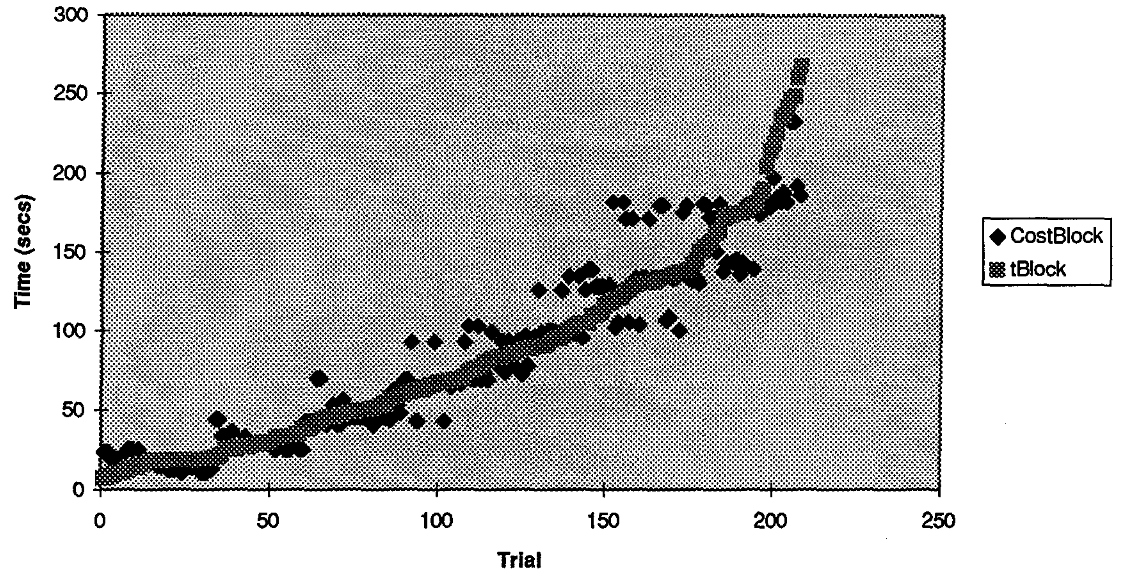


Figure 19 $Cost_{Block}$ **versus** t_{Block}

indicated by the component cost models. The ultimate failure lies with the composite cost models' ($Cost_{Runtime}$ and $Cost_{xRuntime}$) inability to predict the complex, multi-dimensional surface of the runtime versus allocation relationship. While the underlying linear model may be an incorrect form of parameter relationship to runtime, another likely insufficiency is the choice of parameters for the model. Fundamental influences to the performance of VHDL simulations are not included in the current models (e.g. event balance, lookahead, feedback). Furthermore, the simple heuristics implied by a 3-6 term equation cannot accurately predict the pitted surface this and previous research seeks to model. As demonstrated in Figure 21, Kapp's theoretic model also fails to correctly order allocations for just the case of 6 LP simulations of the Wallace Tree multiplier. Unpredictable results can be expected in application to other circuits as well. Note that the models are not necessarily inaccurate, their usefulness is simply limited to an unacceptably small and undefined portion of the search space.

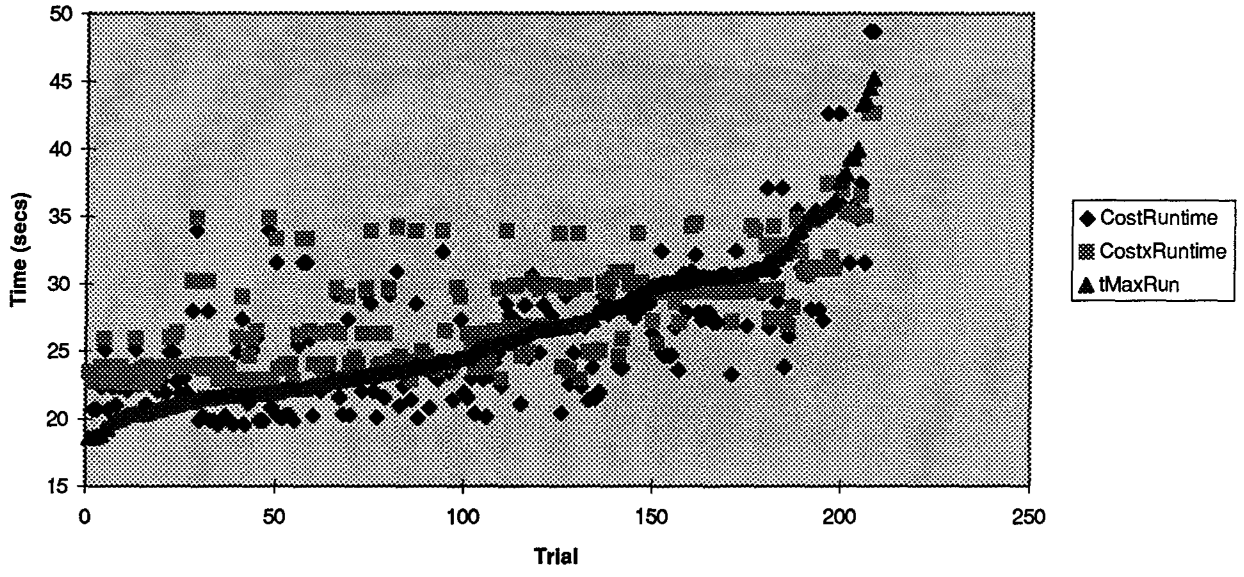
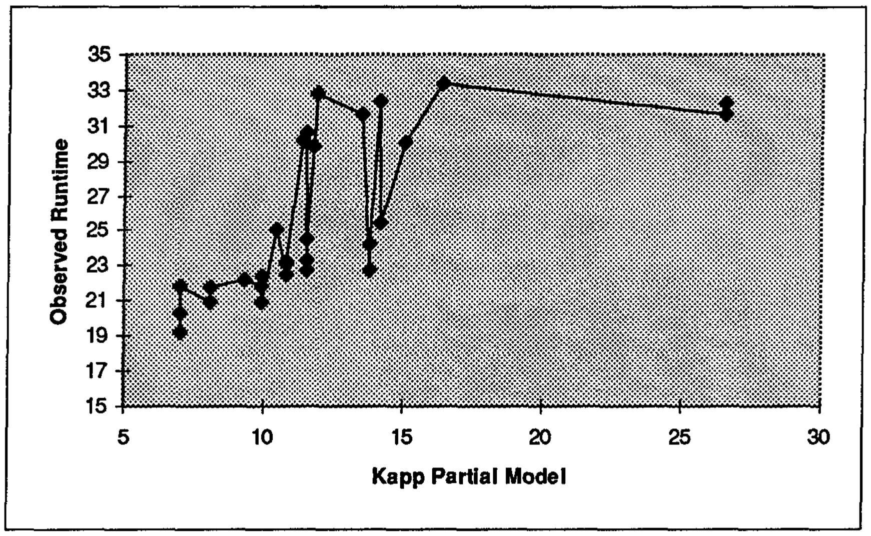


Figure 20 Statistical CostModels versus t_{MaxRun}



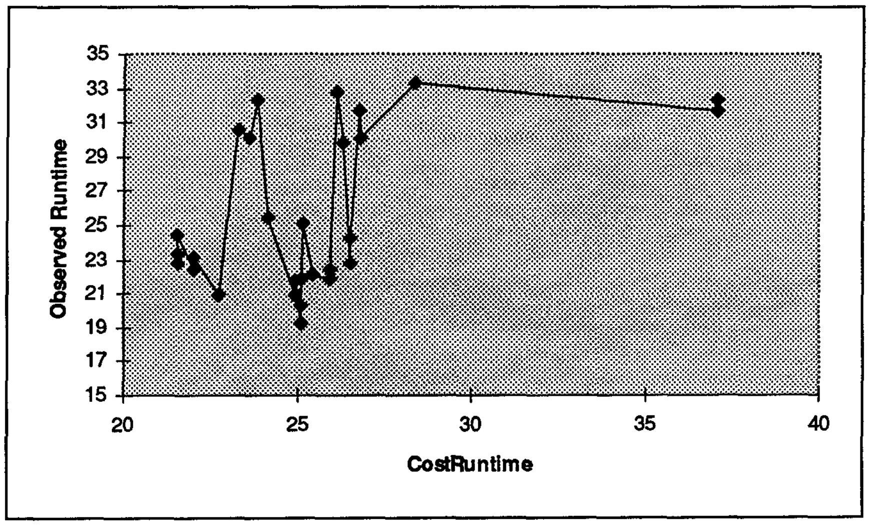


Figure 21 6LP Wallace Tree runtime versus Cost Models

4.3 Model Extrapolation (Assoc Mem)

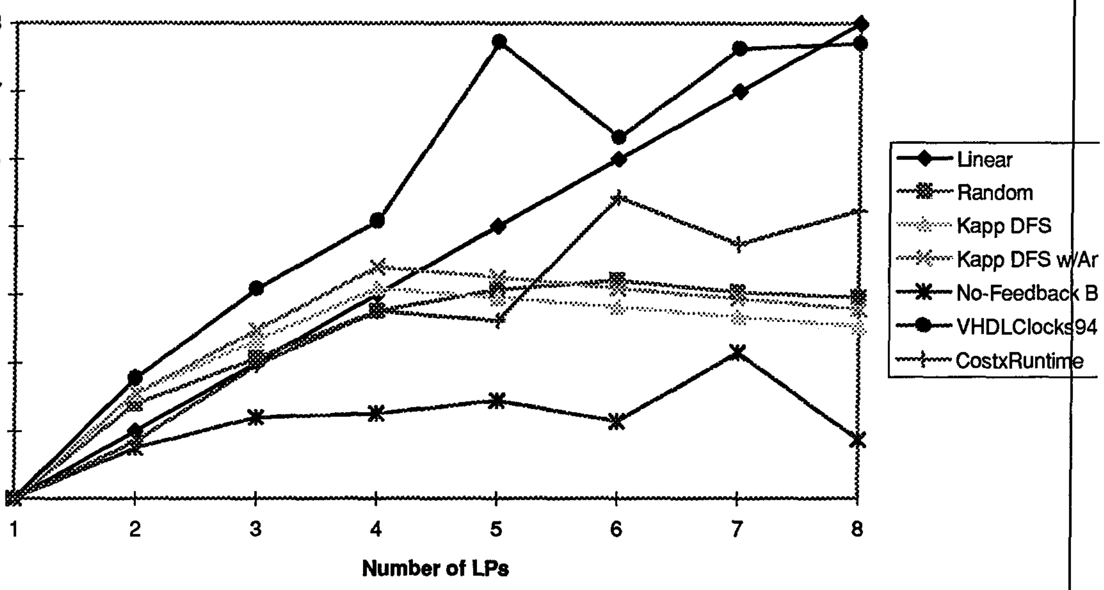


Figure 22 Speedup Curves for Associative Memory

Figure 22 demonstrates the mediocre performance of the cost model for another circuit. Here, however, the $Cost_{xRuntime}$ suffers a handicap since the simple DFS partition has no induced feedback (see next section). Ignorant of the influence of feedback, $Cost_{xRuntime}$ creates the problem. The no-feedback BFS allocation displays the destructive effect of pipeline initialization costs associated with that technique (see next section). The true curiosity of this graph is the uppermost curve which is a simple DFS partition. Simple data partitioning of graphs was enhanced in GPT v3.0. By keeping strongly connected components of the graph together during depth first partitioning, GPT v3.0 coincidentally creates an allocation with no induced feedback. This radically improves the simulation runtime as discussed in the next section.

4.4 Feedback

The effect of feedback on conservative simulations has been qualitatively understood if not quantitatively. Cycles in the problem graph cause participating LPs to proceed in lockstep fashion. No member can proceed until the feedback of its last message propagates around the cycle. Kapp identifies strongly connected components within the problem graph and avoids breaking them over multiple LPs. Contracting the problem graph into strongly connected components results in an

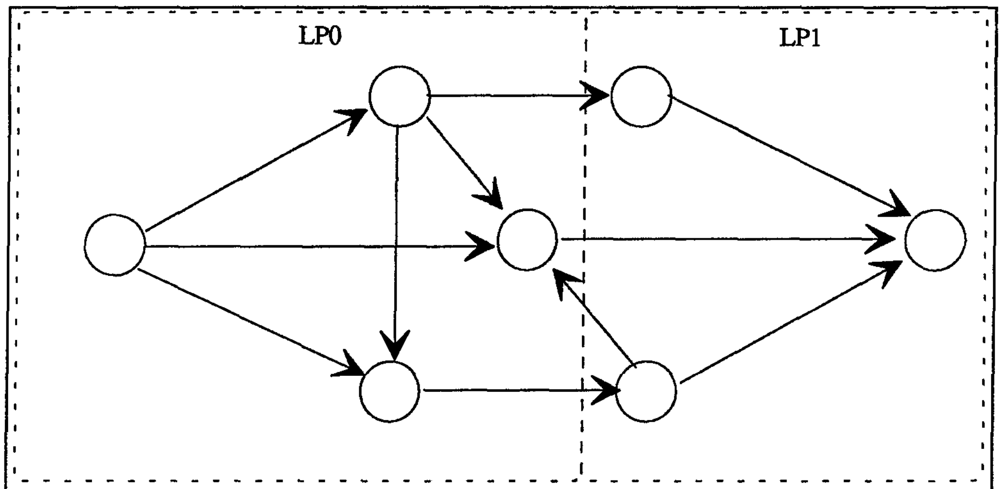


Figure 23 Induced Feedback on Feedforward Network

acyclic directed graph. Another form of feedback is caused by the contraction of the problem graph into the LP graph where it is possible to induce feedback on a feedforward network (Mannix, 1988:3-19). Figure 23 presents an example of *induced feedback*. Oddly enough, Simple Depth First and Simple Breadth First allocations of strongly connected components both fail to guarantee prevention of induced feedback. A general algorithm developed in this research uses the strongly connected component contraction of the problem graph to further contract onto LPs without inducing feedback.

No Feedback Breadth First Allocation

For each vertex (v) define d(v) where
    d(*Source*) = 0 and
    d(v) = max(d(w_i))+1   for all vertices w_i to which v is adjacent - i.e. (w_i,v) is an edge.

Perform a Breadth First Search as

```
Initialize
    color($v$)$\leftarrow$white for all $v$
    $Q \leftarrow$ {all sources}
    $i \leftarrow 0$
while $Q$ is not empty loop
    $v \leftarrow$ head($Q$)
    D $\leftarrow$ d($v$) + 1
    for each $u \in$ Adj($v$) loop
        if color($u$) is white and d($u$) = $D$ then
            color($u$) $\leftarrow$ gray
            Enqueue($Q$,$u$)
    end loop
    Dequeue($Q$)
    $LP_i \leftarrow v$
    if size($LP_i$) > $NumVertices/NumLPs$ then
        $i \leftarrow (i + 1)$
    end if
    color($u$) $\leftarrow$ black
end loop
```

Figure 24 is the resulting partition of applying the algorithm to the graph of Figure 23.

Applying this technique to the Wallace Tree Multiplier resulted in the speedup curve of Figure 25.

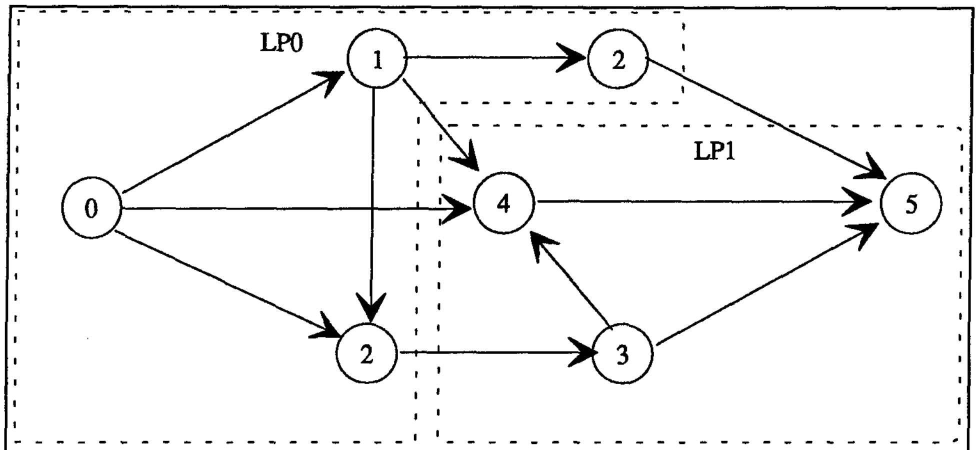


Figure 24 Feedforward Network without Induced Feedback

Superlinear speedup is achieved through a 7 LP partition! As can be seen in Figure 26, the elimination of feedback radically changes the component influences to runtime. Blocking is now a minority influence through 8 processors. The algorithm fails in that it forms an allocation that behaves like a pipelined system. There is an initialization cost to "fill" the pipeline resulting in the end processor having significantly more idle time than the first. This initialization cost can be amortized to insignificance by increasing the number or size of simulation test benches. However, the detrimental effect of "filling the pipe" can be seen in comparing the DFS and BFS No-Feedback partitions of the Associative Memory circuit in Figure 22.

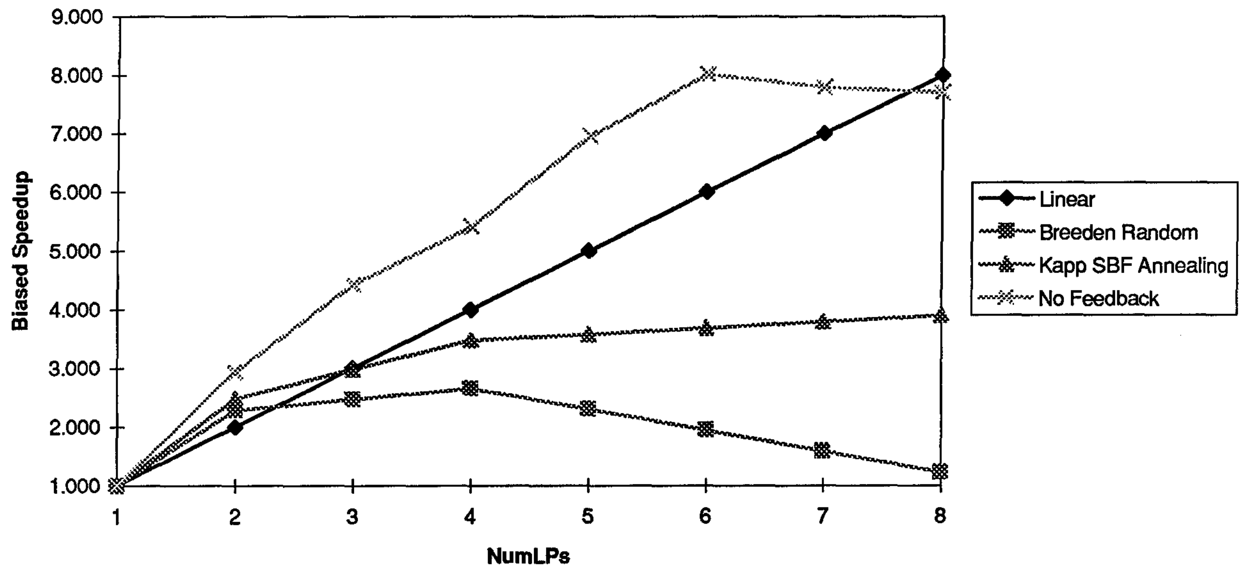


Figure 25 No Feedback Speedup of Wallace Tree

While the identification and exploitation of feedback allows a new, pertinent parameter to enter a cost model, a new cost model must be completely redeveloped due to the drastic changes in the behavior of the circuit. Furthermore, currently there is only a nominal measure of feedback: presence or absence. Feedback may have a graduated influence which would require the definition and measurement of a feedback metric.

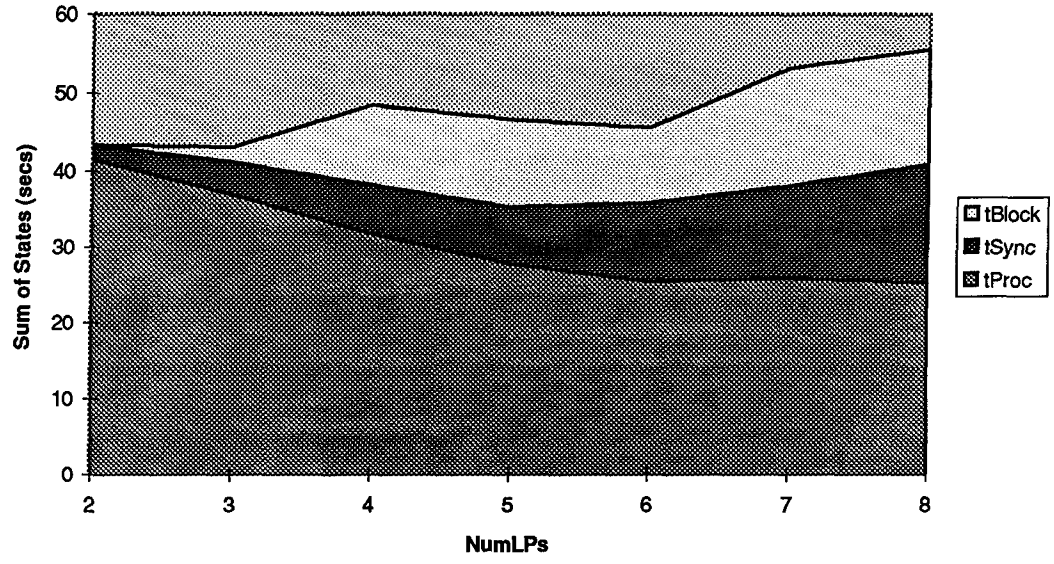


Figure 26 Summary Profile of No Feedback Processing States

5. *Conclusions and Recommendations*

5.1 Research Summary

As modern digital circuits grow larger and more complex, the time required to perform sequential simulations becomes unacceptably slow. Since simulation is a vital input to the design and validation of circuits, this bottleneck affects the efficiency of the entire development cycle. Parallel simulation offers a solution that scales with the problem. By assigning circuit components to distributed processors, the work of the simulation can be divided. There is, however, an additional cost of synchronization between the cooperative processors not present in sequential simulation. The manner in which circuit components are partitioned among processors greatly influences the amount of overhead incurred. The task is to partition intelligently such that computational parallelism is not overwhelmed by synchronization overhead.

In this research effort heuristic techniques of intelligent partitioning were considered. By observing trends of successful partitions, a statistical relationship of *a priori* parameters to parallel simulation runtime was developed. Formal definition of this relationship in the form of a cost model allowed allocations to be ordered by predicted runtime. By choosing the allocation with the lowest cost model value, the simulation using that allocation was expected to have the lowest runtime of the set considered. "The set considered" is an important distinction because the mapping of tasks to processors to achieve the lowest runtime is a known NP-Complete problem. Finding an optimal solution is intractable; finding relatively good solutions is desirable. The set of candidate allocations is chosen by using Kapp's AB Improvement iterative search procedure. This algorithm moves vertices on partition borders to other LPs until moves fail to improve the cost model. The best allocation is then used for simulation.

A 1,050 gate Wallace Tree Multiplier was used to develop and validate the statistical cost models proposed. The 4,243 gate Associative Memory circuit was used to test the general applicability of the cost models.

5.2 Conclusions

The following conclusions about partitioning VHDL circuits for parallel simulation result from this research:

- *Partial ordering of allocations by runtime is still not possible via proposed models.* The relationship that the proposed cost models attempted to emulate is that between runtime and allocation. For any circuit, this surface can be imagined in three dimensions where each allocation is a point in the xy plane and the corresponding observed runtime is plotted along the z-axis. If total knowledge were available, a desired cost model would efficiently follow this surface using a priori parameters. Of course, this target surface has many more dimensions than three. Adjacent allocations differ by the placement of only one behavior. There are $Vertices_{Num}$ available for placement in LPs_{Num}*-1* other LPs, thus the surface has something less than $Vertices_{Num} \bullet (LPs_{Num} -1)$ dimensions. While some dimensions may be constant, it is easy to see the inherent intractability of effectively modeling this complex relationship. As demonstrated in this research, current models fail to partially order allocations under the most restrictive of conditions. Further restrictions would make successful conclusions uninteresting.
- *Good cost models are those that demonstrate stochastic reliability and efficiency, not correct partial ordering.* This paper does **not** suggest the abandonment of simulation cost models. Most intelligent allocations exceed the performance of random partitioning. Instead of attempting to order allocations, cost models that demonstrate good correlation to observed runtime should be pursued. Scatter plots of observed runtime versus cost model results would assist in

determining the form of the model. Correlation and regression measurements would allow quantitative assessment of the model. Computational effort should also be considered in evaluating cost model merit.

- *Statistical model development is an effective technique.* The three component cost models proved very accurate for predicting t_{Proc}, t_{Sync}, and t_{Block} for the validation set. Unfortunately, these predicted parameters do not hold much value. Moreover, the original component cost models will likely fail when applied to no-feedback allocations. But, under the restrictions of feedback-littered simulation of a Wallace Tree Multiplier, the models were accurate. The key, as with any model, is identifying all pertinent parameters and the scope of application. Given this information, statistical model development was able to accommodate and offer insight to the nature of inter-parameter relationships.
- *Feedback is a major contributor to conservative simulation blocking.* The concept of explicit feedback was addressed by Kapp's use of strongly connected components. An efficient algorithm, O($Vertices_{Num}$), is presented as part of this research to guarantee no feedback between LPs. In the case of the Wallace Tree Multiplier, this improvement reduced blocking from an exponential influence (as LPs_{Num} increases) to a linear one. Yet, again, no single parameter is the panacea of VHDL circuit partitioning. Feedback must be measured and included in any effective cost model.
- *Conservative Parallel VHDL Simulation can achieve super-linear speedup.* Speedup curves for both the Wallace Tree Multiplier and the Associative Memory demonstrated speedup to 8 processors (the limit of this research). This verifies that the complexity of VHDL simulation has some form of complexity such that real work reduces as the number of LPs increases (see Figure 16). This implies that scalability is feasible for coarse grain simulation using a conservative protocol.

5.3 Recommendations for Further Research

Given the strong conclusions of this research, several directions should be taken to continue the maturity of conservative simulations at AFIT.

- *Change the research goal from partial ordering to stochastic reliability.* At this evolutionary stage of sophistication for modeling parallel simulation performance, partial ordering is an unrealistic goal. If achieved, the conditions would be so restrictive as to negate the usefulness of the findings. The goal must be redirected to achieving a statistically reliable model in accordance with the conclusion of the previous section. This change in direction would free the researcher to open the scope of investigation since the weight of proof would be limited to statistical evidence, not exhaustive trials of isolated cases. The real burden becomes to collect adequate statistics to form the cost model and demonstrate significant correlation. Perhaps an evolving database of statistics could realize guidelines for model choice based on circuit parameters (e.g. test bench, feedback level, average fanout, etc.).
- *Expand the parameters considered in model development.* As a minimum, the GPT v3.0 needs to maintain LP specific statistics. Valuable insight into the relationship between partitioning and runtime is being lost to summary statistics which hide the **max** function that drives runtime. Many statistics are already gathered and just need to be included in the output.
- *Enhance the search capabilities of GPT v3.0.* Current iterative processing moves a single vertex at a time. This is computational suicide. Evaluation of graph statistics is relatively expensive in the current version of GPT. Moving a single vertex at a time amounts to mowing a soccer field a blade at a time. More drastic steps are required to adequately explore the huge search space. The first method to accomplish this would be to turn the current process into a true annealing routine. It would be better to parallelize a genetic algorithm which can consider

several points at once per processor. However, all techniques will be limited by the current expense of statistics evaluation. Since each iteration of the graph is some delta from the previous cost model value, some technique of marginal evaluation would lower the computational costs and permit more time for searching. Given that this benefit would be realized many thousands of times per search, it should be given significant consideration. Unfortunately, the more complicated the model, the more difficult marginal accounting would be.

- *Continue to enhance the simulation environment.* The first item to change must be the data type of timestamps and clocks in VSIM and SPECTRUM. Overflow has already caused problems for researchers (Kapp, 1993:125). The *time* datatype should be changed to a floating point value. Additionally, the use of t_{VList} as the null message timestamp is extremely limiting. Only when output arcs are the hosts of the next event is that the correct timestamp. If some method of identifying the event host arc were accomplished, a more accurate timestamp would yield the benefit of lookahead.
- *Automate the simulation cycle.* Batch files and utility programs allowed over 600 VHDL simulations to be considered in this research. To support the quantity of results required to provide statistical significance, the entire simulation cycle from partitioning through data collection should be seamlessly automated.

Unfortunately, the objectives of this research were not met. The cost models developed in this study demonstrated no statistical improvement over Kapp's theoretic model. Nor did the models succeed in defining a partial order of allocations for even the limited case of a fixed number of LPs for a single circuit upon which the model was derived. This report concludes that research has not progressed as far as previously thought such that previous findings aren't statistically significant. What was accomplished, however, was the mathematically rigorous investigation and documentation of VHDL simulation behavior. By maturing the tools developed in preceding research, additional instrumentation allowed quantitative exploration of the behavior that limits runtime. It was this instrumentation that demonstrated the decreasing complexity of real work as more processors are utilized in the simulation. Similarly, by eliminating feedback, the dominating effect of conservative blocking was tamed. By introducing statistical methods into this research, elusive theoretic proofs were avoided in favor of stochastic legitimacy.

In a word, this report succeeds in displaying our **ignorance** of simulation behavior. However, by accurately assessing what is known, more productive steps can be taken into the unknown. Statistical techniques are excellent ways to overcome the inability to describe complex systems. Current research of parallel discrete event simulation remains in Thorndike's "organization and prediction" phase of the scientific process. With the appropriate software and mathematical tools in place, maturation to explanation and understanding are just a matter of time.

Bibliography

Bailey, Marl L. and Michael A. Pagels. "Measuring the Overhead in Conservative Parallel Simulations of Multicomputer Programs" Proceedings of the 1991 Winter Simulation Conference: 627-636 (1991).

Breeden, Thomas A. Parallel Simulation of Structural VHDL Circuits on Intel Hypercubes. MS Thesis, AFIT/GCE/ENG/92D-01. School of Electrical and Computer Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1992. DTIC AD-A887491

Chamberlain, Roger D. and Mark A. Franklin. "Hierarchical Discrete-Event Simulation on Hypercube Architectures" IEEE Micro v 10 n 4: 10-20 (Aug 1990).

Chandy, K., and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," Communications of the ACM, vol. 24, April 1981.

Chittor, Suresh and Richard Enbody. "Predicting the Effect of Mapping on the Communication Performance of Large Multicomputers" 1991 International Conference on Parallel Processing: II-1 - II-4 (1991).

Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. Introduction to Algorithms. Cambridge, MA: MIT Press, 1990.

De Vries, Ronald C. "Reducing Null Messages in Misra's Distributed Discrete Event Simulation Method" IEEE Transactions on Software Engineering, Vol 16 Num 1: 82-91 (Jan 1990).

Fujimoto, Richard M. "Performance measurements of distributed simulation strategies" Proceedings of the International Conference on Parallel Processing: 14-20 (1988).

Fujimoto, Richard M. "Lookahead in parallel discrete event simulation" Proceedings of the 1988 International Conference on Parallel Processing: 34-41 (Aug 1988).

Fujimoto, Richard M. "Parallel Discrete Event Simulation" Proceedings of the 1989 Winter Simulation Conference: 1-33 (Dec 1989).

Hennessy, John L. and David A. Patterson. Computer Architecture: a Quantitative Approach. San Mateo CA: Morgan Kaufmann Publishers, Inc., 1990.

Hines, William W. and Douglas C. Montgomery. Probability and Statistics in Engineering and Management Science (Second Edition). New York: John Wiley & Sons, 1980.

Kapp, Kevin L. Partitioning Structural VHDL Circuits for Parallel Execution on Hypercubes. MS Thesis, AFIT/GCE/ENG/93D-07. School of Electrical and Computer Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1993. DTIC AD-A274390

Mannix, David Louis. Distributed Discrete-Event Simulation Using Variants of the Chandy-Misra Algorithm on the Intel Hypercube. MS Thesis, AFIT/GCS/ENG/88D-14. School of Electrical and Computer Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988. DTIC AD-A202849

Mathcad Plus 5.0 for Windows. 80386 or better IBM Compatible, 8 Mb RAM, MS Windows 3.1 or later. Computer Software. MathSoft Inc, Cambridge. MA, 1994.

Nandy, Biswajit and Wayne M. Loucks. "On a Parallel Partitioning Technique for use with Conservative Parallel Simulation" Proceedings of the 1993 Workshop on Parallel and Distributed Simulation: 43-51 (May 1993).

Nicol, David M. "Performance Bounds on Parallel Self-Initiating Discrete-Event Simulations" ACM Transactions on Modeling and Computer Simulations, v 1 n 1: 24-50 (1991).

Nicol, David M. "The Cost of Conservative Synchronization in Parallel Discrete Event Simulations" Journal of the ACM, v 40 n 2: 304-333 (Apr 1993).

Sartor, JoAnn M. Optimal Iterative Task Scheduling for Parallel Simulations. MS Thesis, AFIT/GCS/ENG/91M-03. School of Electrical and Computer Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, May 1991. DTIC AD-A238631

Soule Larry and Anoop Gupta. Characterization of Parallelism and Deadlocks in Distributed Digital Logic Simulation" Proceedings of the 26th ACM/IEEE Design Automation Conference: 81-86 (1989).

Thorndike, Robert M. Correlational Procedures for Research. New York: Gardner Press Inc, 1978.

Wagner, David B. and Edward D. Lazowska. "Parallel Simulation of Queueing Networks: Limitations and Potentials" Performance Evaluation Review, v 17 n 1: 146-155 (May 1989).

Weiland, Frederick, Peter Reiher, and David Jefferson. "Speedup Bias" California Institute of Technology (1990).

Wilson, Robin J., Introduction to Graph Theory (Third Edition). Essex, England: Longman Scientific and Technical, 1985.

Vita

PII Redacted

Joel F. Hurford was [redacted]. In June, 1985 he graduated from Rochester Adams High School, just one month before reporting at the US Air Force Academy. There he earned a Bachelor of Science degree in Computer Science and was commissioned a second lieutenant in the US Air Force in May, 1989. His first assignment was with the 7th Communications Group, Pentagon, Washington D.C.. Working as a System Administrator, Database Administrator, and Network Administrator, he directly supported the Offices of the Air Staff. In May 1993, Capt Hurford began his pursuit of a Master of Science in Computer Engineering at AFIT.

Permanent Address: [redacted]

PII Redacted

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE Accelerating Conservative Parallel Simulation of VHDL Circuits	5. FUNDING NUMBERS
6. AUTHOR(S) Joel Hurford	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-6583	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/94D-10
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Mr. Robert Parker ARPA/CSTO 3701 N. Fairfax Dr. Arlington, VA 22203	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Distribution Unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This research effort considers heuristic and cost model based techniques for the optimal partitioning of VHDL circuits for parallel simulation. Correlation statistics are gathered on a wide variety of graph-based *a priori* parameters. Linear regression is used to identify significant parameters for inclusion in a representative cost model. Driving a greedy search, this cost model is used to improve upon initial heuristic partitions. The influence of feedback dominated previous research so a no-feedback algorithm is used to create the initial partition. The circuits studied range from 1,050 to 4,243 gates.

14. SUBJECT TERMS Parallel Discrete Event Simulation, Conservative, VHDL, Feedback Static Dependency Graph, superlinear	15. NUMBER OF PAGES 99
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL